AFRL-RI-RS-TR-2009-99
Final Technical Report
April 2009

NETWORK AUTHENTICATION PROTOCOL STUDIES

University of Idaho

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. N665/03

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2009-99 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/ /s/

W. JOHN MAXEY
Work Unit Manager

WARREN H. DEBANY, Jr.
Technical Advisor, Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* APR 09	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* Jul 02 – Jun 08

4. TITLE AND SUBTITLE NETWORK AUTHENTICATION PROTOCOL STUDIES	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER F30602-02-1-0178
	5c. PROGRAM ELEMENT NUMBER 62301E
6. AUTHOR(S) Jim Alves-Foss and Paul Oman	**5d. PROJECT NUMBER** N665
	5e. TASK NUMBER A6
	5f. WORK UNIT NUMBER 10
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Idaho, Admin Annex Financial Affairs, Moscow ID 83844-0001	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency, 3701 North Fairfax Drive, Arlington VA 22203-1714; AFRL/RIGB, 525 Brooks Rd., Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2009-99

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# AFMC 2008-0701

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The original focus of this project was to investigate cryptographic protocols, and methods for formal design and analysis of those protocols. As time progressed, we found a need to broaden the scope of this work to include the foundational system architecture support needed for these protocols. In this report we provide a summary of the work we conducted during this study, the findings and a proposed path forward.

15. SUBJECT TERMS
Network Authentication Protocols, Multi-Level Security, Security Policy

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** William J. Maxey
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	216	**19b. TELEPHONE NUMBER (*Include area code*)** N/A

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Contents

LIST OF FIGURES

LIST OF TABLES

Executive Summary

The original focus of this project was to investigate cryptographic protocols, and methods for formal design and analysis of those protocols. As time progressed, we found a need to broaden the scope of this work to include the foundational system architecture support needed for these protocols. In this report we provide a summary of the work we conducted during this study, the findings and a proposed path forward.

Cryptographic protocols are network communication protocols developed for the exchange of information between communicating parties to enable the establishment of one-way or mutual authentication, shared encryption keys or establishment in trust of public keys. The strength of these protocols is based in their use of cryptographic technologies, hence the term cryptographic protocols.

Most cryptographic protocols are relatively simple, consisting of the exchange of a few simple messages to no more than a few hundred. Compared to the complexity of the software using them, often hundreds of thousands or millions of lines of code, they should be much easier to understand, analyze and use. It is therefore somewhat surprising to find numerous examples of flaws in the protocols in the literature.

In our work, presented here, we examined the basis of these flaws, as an extension of our early work. We found that the flaws often manifest themselves in a misunderstanding of the full semantic meaning of the protocols. This leads to numerous types of attacks, including type-based attacks, guessing attacks and generation of new protocols to allow for tailored attacks. We developed a new group-based key exchange protocol, CCEGK, built on top of simple two-party key exchange protocols to examine a higher-level application of cryptographic protocols, in an attempt to see if the hierarchy presented good performance and simpler verification and assurance.

We found that the flaws found in these protocols occur when an untrusted party is able to violate some of the inherent assumptions of protocols designers. For example, an attacker can replay portions of prior messages in place of new ones, can substitute values, can interleave messages from concurrent protocols, or can attack simple passwords used to protect strong encryption keys. In our design of CCEGK, we built a hierarchical protocol structure to attempt to utilize strong lower levels of encryption and authentication to build a more complex higher level protocol. We also began to examine the use of CORBA message passing in an attempt to understand application-level communication that would utilize lower level security protocols.

From this examination of hierarchies and protocol layering, we found a need for a trusted platform on which to build the cryptographic protocols. If the underlying cryptographic primitives can not be trusted and protected, then our protocols will always be insecure. If a user could install

new software to create tailored protocols attacks, our protocols will not be secure.

Based on this observation, we moved to the next, much larger phase of this project, the development of the Multiple Independent Levels of Security Architecture (MILS). The intent of the MILS architecture is to develop a solid foundation for the design and analysis of high assurance computing systems. The initial application will be secure communication system. In a MILS system, security relevant components are isolated in separate execution environments (partitions), where we are guaranteed data isolation, controlled information flow between partitions, fault and attack containment, and easier verification.

This report is organized into three main parts. Part I provides background and terminology related to the concept of network protocols design and analysis and summarizes the main results of this portion of the project, focusing on the network protocol analysis and the design of the CCEGK protocol. Part II provides and overview of the MILS work. Part IV provides a bibliography of the cited research and then the abstracts of the published worked generated from this project.

Part I

Network Authentication Protocols

Overview of Part I

The first part of this report summarizes the concerns related to the design and analysis of network authentication protocols, specifically types of attacks that can be launched against these protocols and then the research we conducted in this area. Specifically we first address the formal analysis of network authentication protocols. Then we address the design and analysis of a new group key agreement protocol.

The work highlighted in this section has been published in several papers, and results in masters thesis and dissertations. The abstracts for these publications can be found in final section of this report. Publications related to the analysis of network authentication protocols are filed under the following keys: Corin03a [86], Malladi02a [174], Malladi02b [172], Malladi02c [173], Malladi02d [169], Malladi03a [171], and Malladi04a [170]. Publications related to the generation of the new group key agreement protocol are filed under the following keys: Manz05a [176], Manz07a [177], Zheng05a [284], Zheng06a [280], Zheng06b [283], Zheng06c [281], Zheng07a [282].

In addition, we did development and experiments with some other security protocols for secure distributed firewalls, secure mobile agents, intrusion detection system message passing, remotely accessible MLS file server, and network filters/guards for protocols as reported in: Al-Muhaitheef02a [24], Lee02a [150], Lee04a [151], Meyers07a [192], Robinson07a [221], Robinson07b [222], Robinson07c [223], Rossebo06a [227], and Yang05d [274].

Chapter 1

Introduction

1.1 Security Protocols

A *protocol* comprises of a prescribed sequence of interactions between entities designed to achieve a certain end. A communications protocol is designed to establish communication between agents, i.e setup a link, agree on syntax, etc. Simple things like banking transactions and email make use of protocols. The goals of security protocols, (sometimes known as cryptographic protocols), involve providing security services in a distributed system. Some of its goals are : to establish a secret key between two entities; authentication of one entity to another; and ensuring secrecy, integrity, anonymity, non-repudiation etc. Often, they involve the exchange of messages between nodes, requiring a trusted third party (eg. a session server). They may use different cryptographic mechanisms like symmetric or asymmetric encryption, public key cryptosystems, hashes, digital signatures and digital certificates. Obviously, these protocols have to undergo a lot of design and analysis. These tasks are not trivial, and many difficulties can arise:

- The environment in which security protocols operate is very complex. Each assumption about the environment must be defined explicitly. However, not all vendors will follow these design rules, and worse yet, malicious users likely will never follow these rules [28].
- The goals to be achieved and the properties that have to be ensured are very subtle. Simple notions such as authentication and non-repudiation are beset with lot of subtleties. A hot debate in this field revolves around attributing a precise meaning to these concepts.
- Listing all of the capabilities of the penetrator is very difficult. (We shall call a hostile agent in the communication a penetrator. Spy, malicious netizen, eavesdropper, attacker etc. are also common terms for penetrator in the literature. We shall also use intruder synonymously with penetrator. Although it is impractical to list out all the capabilities of a penetrator, we will try to make good approximations.
- Security protocols also involve a high degree of concurrency. This makes the analysis even more challenging.

Given these factors, security protocols are candidates for rigorous design techniques and exhaustive analysis techniques. In the recent past, especially in the 90's, a lot of effort was put into the analysis of security protocols. These included approaches that made use of formal methods (model checking and theorem proving) and some other approaches. We will later explain those approaches in detail.

1.2 Security Properties

First, we shall list out the security properties usually required to be preserved by security protocols. All security protocols do not attempt to provide all of these properties. Usually they are designed to provide a subset of these properties. These properties are interpreted differently by different sources. Therefore, it is important to explicitly define the properties in the context of a protocol. For example, correctness of a protocol can mean many things in many contexts. If it is an authentication protocol, correctness might mean both the participants being authenticated to each other properly. But it would not mean establishing a secret key for communication between two participants without a penetrator learning the secret. Literature is replete with attacks on security protocols due to misinterpretation of security properties that the protocols provide and goals that they need to achieve.

1.2.1 Secrecy

Secrecy [136] can have different meanings in different contexts. For example, a certain application might require that a penetrator should not be able to know anything from a communication between honest agents. This can be regarded as the highest level of secrecy, so much that the penetrator must not be able to do any traffic analysis. This obviously is hard to achieve except through some designs in the physical communication layer that prevent anyone from doing traffic monitoring. Usually security protocols do not need this level of secrecy. In this thesis, we will assume that it is sufficient to prevent the penetrator from obtaining any information that is intended to remain secret through an analysis of the traffic, as in [102]. We will not be concerned that a penetrator is able to obtain encrypted information through traffic monitoring, as long as he doesn't get the corresponding decrypting key.

1.2.2 Authentication

We shall discuss two types of authentication here.

1. Authentication of origin
2. Entity Authentication.

Authentication of origin is satisfied if a message is thought of as being sent by a party, and it was indeed sent by that party. It would sometimes require that we can be sure that the message was not modified or tampered before it reached it's destination. Informally, we can put it this way. Authentication is maintained in a communication between Alice and Bob if in the communication whenever Bob accepts a message as being sent by Alice, Alice indeed has sent it. A stronger form of this would be to extend it further and require that Alice intended to send the message for Bob. Gavin Lowe defines authentication in [163], as:

> "Whenever an agent A completes a run in the protocol, apparently with B, then B has recently been running the protocol, apparently with A, and the two agents agree upon who initiated the run, and agree upon all data values used in run; further, there is a one-to-one relationship between the runs of A and the runs of B."

The distinction between entity authentication and authentication of origin is not clear in the literature. In many places they are used synonymously. The confusion mainly originates from the fact that some designers assume connection oriented communication and some assume connection less. Intuitively, entity authentication means that the claimed identity of an entity to another party in a communication is correct. Menezes et al. [191] define entity authentication as

> "the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e., is active at, or immediately prior to, the time the evidence is acquired)."

1.2.3 Non-Repudiation

Non-repudiation is similar to authentication but it has stronger requirements for proof. The major distinction in authentication arises from the fact that while authentication requires that a party show proof of it's identity to the other party in communication, non-repudiation requires that the proof should also be demonstrated to a third party, validating the truth in the claim [286]. Non-repudiation services provide protection to parties involved in a transaction against the other party denying involvement in an action or event subsequent to the happening of the action or event. Two international standards have been proposed to deal with non-repudiation. ISO/IEC 10181-4 [19] and ISO/IEC 13888 [20, 22, 21]. While ISO/IEC 10181-4 provides a framework for developing and employing these services, ISO/IEC 13888 consists of three parts. They propose a general model of non-repudiation and a set of non-repudiation mechanisms based on cryptography (both symmetric and asymmetric cryptographic techniques). Three security elements for non-repudiation have been defined in CCITT X.400 series recommendation [273]. They are:

- *Non-repudiation of Origin*: The receiving party of a message in a communication is provided with evidence of message origination and is protected against subsequent denial by the originator.
- *Non-repudiation of Delivery*: The sending party in a communication is provided with evidence of message delivery to the other party and is protected against subsequent denial by the receiving party.
- *Non-repudiation of Submission (NRS)*: The sending party of a message in a communication is provided with proof of submission that the delivery agent has indeed submitted the message to the originally specified recipient(s) and is protected against subsequent denial by the delivery agent.

1.2.4 Anonymity

Anonymity [234] is a property that has not been explored as much as other security properties from a formal perspective. A system anonymous over a set of events E should have the property that if an event occurs from the set E, an observer, though he might be able to deduce that the event occurred, wouldn't be able to identify the event. In a typical situation, the set E will be an indexing set of users, so that an observer might be able to tell that an event from set E has occurred but he will not able to judge which of the agents was responsible for the event's occurrence. Anonymity need not necessarily apply to each occurrence of events. In fact, they can be reshuffled upon each occurrence of an event. We might not expect in some situations that the occurrence of events from the set will be wholly independent. A good example would be voting. In this case, one would like a particular vote to be associated with an individual. However, we might apply a restriction of no double voting, so the events now are not independent but have the constraint on them that for a given run, each event should occur at most once.

1.3 Cryptography

We shall treat cryptographic algorithms in a very abstract way in this thesis. Cryptographic algorithms are a different direction of research in the field of security protocol analysis. Researchers in these fields attend separate conferences and have separate journals etc. In this thesis, we will assume that cryptographic algorithms that are used in the security protocols are safe. This assumption forms one of the main assumptions that any protocol analyzer would generally make. As in [28], we assume that:

1. It is computationally infeasible to break the underlying cryptographic algorithm. Otherwise, the whole protocol would be deemed as insecure.
2. Also, we assume that there exist secure cryptographic modules that perform as specified and do not contain any *Trojan horses* and there exist adequate operating system security measures such that the stored copies of user's keys can not be compromised; otherwise flaws in these systems can be used to subvert the protocol.

Cryptographic algorithm analysis and protocol design and analysis are now made completely independent by researchers. This is done partly to keep the analysis tractable. i.e., an analysis that involves both the aspects become extremely difficult to complete and can become practically intractable. These fields are separated into two different communities, the cryptologists and the formal analysts. Although these communities have been largely disjoint, they are beginning to collaborate with interchanging ideas and techniques of one field to the other. In this section, we shall describe the general cryptographic techniques that are in use today, *symmetric key cryptography* and *public key cryptography*. We shall also introduce some famous cryptographic algorithms and key exchanging routines that are commonly used to create and exchange keys (RSA algorithm, Diffie-Hellman key exchange etc.). Lastly, we shall define and describe a common term: nonces.

Cryptography is traditionally described in relation to *messages*. The original message is called the *plain-text*. The message obtained after applying cryptography is called *cipher-text*. The process

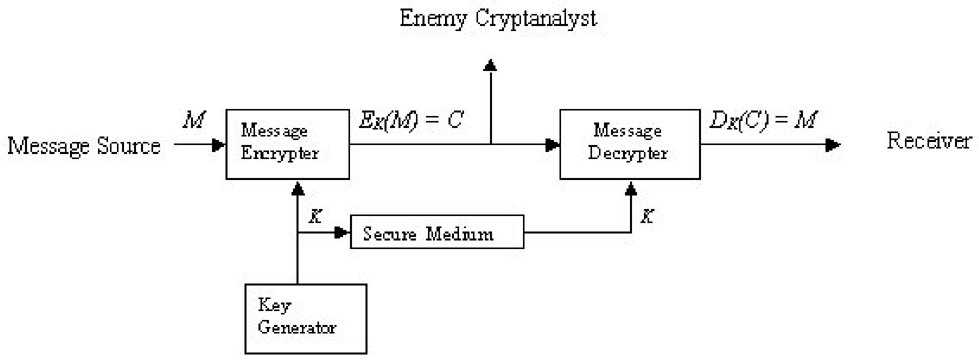

Figure 1.1: A Cryptosystem

is called *encipherment* or *encryption.*[1] The process that does the reverse of encryption, i.e. transforming a cipher-text into plain-text is called *deciphering* or *decryption.* So, if M is a plain-text message, E and D the transformations for encryption and decryption explained above, then,

$$D(E(M)) = M$$

The transformation E (for encryption) has usually some parameters called *keys*. Let K be the set of keys. Then we say,

$$C = E(M, K)$$

if C is the cipher-text obtained by encrypting message M with key $K \in \mathsf{K}$. A *cryptosystem* is the entire set-up (the transformations E, D and keys K).

1.3.1 Symmetric Cryptography

Shannon first presented the theory of classical cryptosystems [240]. *Classical* (or *conventional*) cryptosystems use symmetric key cryptography to encrypt and decrypt messages. They transform a message so that a message that is encrypted with a secret key can only be read by a person possessing the secret key for that transformation. If E and D represent the encryption and decryption functions respectively, and K is the secret key for the transformation, then, the following must hold:

$$D(E(M)) = M.$$

Let C be the cipher-text created by encrypting message M by the transformation E using key K. This is represented as,

$$C = E_K(M).$$

Also, decryption of C using the same key K is represented as,

$$D_K(C) = M$$

and this must be true for a classical cryptosystem. Below, we present some techniques used in classical cryptosystem [250].

[1]Thus we will call *cipher-text*, *encrypted text*, *encrypted message* or *encrypted component.*

Substitution

These can be classified into *monoalphabetic* and *polyalphabetic* substitutions. A *simple substitution cipher* replaces each letter in a message stream with a substitute. For example, consider the following permutation of the alphabet:

WQYUIOPGFDSAHJKLVCXZBNMTRE

When a message is sent, each letter in the message is substituted with the corresponding substitute in the key. For example, the word "ATTACK" becomes "WZZWYS". Simple substitutions have been proven to be vulnerable to statistical analysis.

A *polyalphabetic cipher* uses several different permuted alphabets in the key, following in a periodic way. With five alphabets, for example, the first alphabet would encipher 1, 5, 6, 7, and 8. The second alphabet enciphers 2, 3, 9, 14, and 15 and so on. Polyalphabetic ciphers were solved way back in the nineteenth century by a method for calculating the period. A cryptanalyst who discovers this period can then decipher the message by splitting it into several *monoalphabetic* ciphers.

Transposition

Transposition ciphers are obtained by splitting the original message into fixed length blocks and arranging the letters inside each block according to some preset permutation.

For example, if the original message is

Secur|ity of| crypt|osyst|em too

For a block length of 5 and permutation (2, 5, 4, 1, 3), we get

$$Erusc|tfoiy|rotpcy|stsoy|mooet$$

as the cipher-text.

Transposition of messages hides the statistical properties of letter pairs such as "of, too" etc. Transposition ciphers are attacked using frequency tables for the combinations and by finding permutations that bring back the original combinations. Substitution and transposition ciphers form the most common forms of cryptosystems. There are several cryptosystems like *Vigenère ciphers, Running-key ciphers, Vernam ciphers* etc.

Block and Stream Ciphers

Substitution ciphers are further divided into *block* and *stream ciphers* [250]. A block cipher divides the plaintext into blocks of fixed length and applies the same encryption function (same encryption algorithm and key) to each block. For example, if the blocks are $M_1, M_2, M_3, \ldots,$

Then,

$$E_K(M) = E_K(M_1)E_K(M_2)\ldots$$

In other words, the cipher-text thus obtained would be a concatenation of ciphertexts from such individual block ciphers, obtained by applying the same encryption transformation to each block. Intuitively, a substitution cipher is a block cipher whose block length is equal to one.

Stream ciphers are obtained by applying encryption to each individual digit in the message stream. A *key stream generator* which generates a sequence of digits called the *key stream* is used in this cryptosystem. The digits in the message and key stream are exclusive-ored to form bits in the cipher-text.

Thus, each bit C_n of the cipher stream is formed as $C_n = M_n \oplus K_n'$, where M_n is the n^{th} bit of the plain-text and K_n is the n^{th} bit of the *key-stream.* Similarly, the plain text is obtained by doing *modulo* 2 addition between the bits of the key stream and bits of the cipher-text. Thus, M_n is obtained by $M_n = C_n \oplus K_n$. Stream ciphers have the advantage that the encryption of plain text and decryption of cipher-text can be done using the same cryptosystem. However, the difficulty and the challenge lies in designing a generator for generating the key stream bits.

1.3.2 Asymmetric or Public-Key Cryptography

In conventional cryptosystems, secret information will be shared by the sender and receiver. *Public key systems* are designed in such a way that they contain both public and private information. They are not only used for key distribution, but also for a much more important cause called message authentication.

Diffie-Hellman Concepts in Public-Key Cryptosystems

Diffie and Hellman (1976) introduced the idea of public key cryptosystems [94]. In the Diffie and Hellman public-key cryptosystem, there are two transformations, namely, E_K and D_K. They both stand for encryption and decryption transformations respectively. The properties of the cryptosystem are [250]:

1. For every key K in the cryptosystem,

   $$D_K(E_K(M)) = M$$

   i.e. decrypting an encrypted message yields the same message.
2. The transformations E_K and D_K are easy to compute.
3. It is computationally infeasible to know D_K from E_K for almost every K.
4. Encrypting a decrypted message also yields the same message. I.e.

   $$E_K(D_K(M)) = M$$

5. It is feasible to compute E_K and D_K for every key K in the cryptosystem.

From property 3, D cannot be constructed from E, so it is perfectly secure for E to be publicly known.

A public key system works in this way: *Alice* generates a pair of inverse transformations, E_A and D_A. *Alice* makes the encrypting transformation publicly known. The decrypting transformation is kept secret. E_A is *Alice's* public key, D_K is the private key.

When *Bob* wishes to send a message M to *Alice* so that only *Alice* can read it, *Bob* :

1. Looks up in the public key directory to find *Alice's* public key, E_A;
2. Uses that key to generate an encrypted message, $E_A(M)$;
3. Sends this encrypted message to *Alice*.

Upon receiving $E_A(M)$, *Alice* applies the secret transformation to obtain the plaintext message.

$$D_K(E_K(M)) = M$$

Diffie-Hellman Algorithm Overview:

Diffie and Hellman went on to propose a way to find the transformations, E and D. We shall give an overview of the algorithm here [250].

The protocol has two system parameters p and g. They are both public and may be used by all the users in a system. Parameter p is a prime number and g is an integer less than p such that there exists an exponent k, where for every number n between 1 and $p-1, n = g^k \bmod p$.

The Diffie-Hellman Public key distribution system works as follows:

- Assume *Alice* and *Bob* want to exchange a secret key among themselves over an insecure channel.
- *Alice* generates a random private value a and *Bob* generates a random private value b. Both a and b are chosen from the set of integers $\{1, \ldots, p-2\}$.
- Then, both compute the following using their public values g and p and their private values a and b.
- *Alice* computes, $g^a \bmod p$ and *Bob* computes $g^b \bmod p$. These are their respective public keys. They exchange their public values.
- Finally, Alice computes $g^{ab} = (g^b)^a \bmod p$ and Bob computes $g^{ba} = (g^a)^b \bmod p$. Because, $g^{ab} = g^{ba} = k$, Alice and Bob now have a shared secret key k.

The method is based on exponentiation in a *finite* (*Galois*) field over integers modulo a prime, or a polynomial field. It's security relies on the difficulty of computing the logarithms in these fields.

1.4 Examples of Security Protocols

Having introduced security protocols and cryptography, we will now present the structure of a security protocol and two example protocols to which we will frequently refer in this thesis. A security protocol consists of a set of a finite number of sent and received messages between finite number of participants. It is represented as a sequence of message transmissions in the as

$$A \rightarrow B : M.$$

Here, A and B are the *agents* or *participants* in the protocol. M represents a message (unencrypted, encrypted, or a combination of both). So, the expression represents agent A, sending message M to agent B. We term this a protocol rule. A protocol is a sequence of such rules numbered from 1 to n, where n is the total number of rules in the protocol.

Let us look at the adapted Needham-Schroeder Public-Key Protocol (NSPK) [162]:

Msg 1. $a \rightarrow s : b$
Msg 2. $s \rightarrow a : \{PK(b), b\}_{SK(s)}$
Msg 3. $a \rightarrow b : \{n_a, a\}_{PK(b)}$
Msg 4. $b \rightarrow s : a$
Msg 5. $s \rightarrow b : \{PK(a), a\}_{SK(s)}$
Msg 6. $b \rightarrow a : \{n_a, n_b, b\}_{PK(a)}$
Msg 7. $a \rightarrow b : \{n_b\}_{PK(b)}$.

Before explaining this protocol, let us introduce some standard notations that we will follow using this example: a and b are agent *identities* (or *roles*). There is a distinction between roles and the actual agents participating in the protocol. We shall use participants' names like *Alice* and *Bob* to represent participants playing the roles in the protocol. s is the identity of a trusted server; n_a and n_b are nonces (explained in the next section); the functions PK and SK return an agent's public key and secret key respectively; $\{m\}_k$ denotes m encrypted by key k. Also, the above variables are considered to be free variables that can be instantiated with different values in different runs.

Let us explain this protocol, step-by-step.

Msg 1. $\mathbf{a \rightarrow s : b}$

Agent 'a' sends to server the identity of agent 'b'.

Msg 2. $\mathbf{s \rightarrow a : \{PK(b), b\}_{SK(s,a)}}$

Server responds and sends to a the public key of b and identity of b, encrypted with it's session key / shared key with a; thus, a can now have the public key of b.

Msg 3. $\mathbf{a \rightarrow b : \{n_a, a\}_{PK(b)}}$

a sends b, a random nonce n_a (created by a), together with it's identity encrypted with b's public key (obtained in step 2).

Msg 4. $\mathbf{b \rightarrow s : a}$

b then sends the server the identity of a.

Msg 5. $\mathbf{s \rightarrow b : \{PK(a), a\}_{SK(s,b)}}$

The server repeats the same as in step 2. But this time it sends to b the public key of a and identity of a encrypted with the key it shares with b.

Msg 6. $\mathbf{b \rightarrow a : \{n_a, n_b, b\}_{PK(a)}}$

b now sends to a n_a and n_b together with it's identity, encrypted with public key of a (obtained in step 5). Now only a can decrypt this message.

Msg 7. $\mathbf{a \rightarrow b : \{n_b\}_{PK(b)}}$

Lastly, a sends back to b, n_b (obtained by decrypting message in step 6 with it's private key) encrypted with b's public key; thus, only b can read this value (n_b). Also, b can check this value with the one it sent to a so that b is convinced that a has created this message and no one else. n_b now becomes the secret key or shared key for a and b for subsequent communications between them.

As another example, consider the Woo and Lam protocol from [272].

Msg 1. $a \rightarrow b : a$
Msg 2. $b \rightarrow a : n_b$
Msg 3. $a \rightarrow b : \{a, b, n_b\}_{shared(a,s)}$
Msg 4. $b \rightarrow s : \{a, b, \{a, b, n_b\}_{shared(a,s)}\}_{shared(b,s)}$
Msg 5. $s \rightarrow b : \{a, b, n_b\}_{shared(b,s)}$

Note that b cannot decrypt the message he receives in message 3, but instead simply includes it inside message 4.

The first protocol is a classical example of a protocol that satisfies the property of *secrecy*. The second protocol satisfies the property of *authentication*. (Observe that there is no shared value established between agents in the Woo and Lam protocol. The ultimate goal of the protocol is that b should believe that a is really whom it claims to be). We will use these protocols as examples in several places to illustrate our concepts and definitions.

1.4.1 Nonces

The definition of a *nonce* can be informally taken to be a fresh, random value. The dictionary definition of a nonce however, reflects the phrases, 'time being', or 'present occasion'. A 'nonce-word' is used in place of 'one occasion'.

In cryptographic protocol design, it's use is subject to it's creation—it has to be created such that it is guaranteed that it is unpredictable and unique. The exact properties however, vary depending on how they are used exactly. The key idea in using nonces is to establish causal relationship between messages. For example, consider the Needham-Schroeder Public-key Protocol presented above. In the third message (Msg 3), Alice sends Bob the message, $\{n_a, a\}_{PK(b)}$. If Alice didn't include the nonce n_a inside the encryption, she is open to an attack as follows: intruder Yves, masquerading as Bob, can create the message 6 as $\{n_Y, b\}_{PK(A)}$ and send it to Alice. Since Alice has no way of knowing that this message was created by somebody other than Bob, she innocently sends $\{n_b\}_{PK(b)}$ as the seventh message. Now, n_Y becomes the secret key between 'Bob' and Alice.

Yves can subsequently obstruct all messages sent to Bob by Alice and read them with the help of this secret key. The attack works because of the lack of causality between msg 3 and msg 6. With the help of a nonce in message 3, Alice can check the value sent in message 6 with the one she generated to know that Bob was indeed able to decrypt message 3. Other uses of nonces also exist. For example, many times they are used to prevent replay attacks. Since they are created to be fresh and unique, replay of encrypted components that contain them in subsequent messages were thought to be easy to detect. However, as we will see later, this is not always the case. We will show that nonces cannot prevent all types of replay attacks, especially those that work even in the presence of freshly created, random, unique values. The implementation of nonces is done using freshly created random values drawn from some large space of possible values. With bit strings of sufficiently long length (say 128), it is possible to make the possibility of a nonce failing to be unique negligible.

1.5 Attacks

Having briefly introduced security protocols, security properties and cryptosystems used by security protocols, we shall discuss attacks on security protocols in this section. Attacks on security protocols are defined as a breach of security properties of the protocols. As said before, all the protocols need not have all the security properties (secrecy, authentication, non-repudiation etc.). Each protocol could be designed to satisfy just a subset of those properties. Any violation of those properties by a penetrator using some technique(s) without the notice of honest agents is an attack on the protocol. Many attacks continue to be published on security protocols (sometimes years after they are first proposed [162, 163, 133]). Analysts continue to find a variety of attacks on protocols through their analysis techniques. Even the protocols presented in previous section (NS protocol, Woo and Lam protocol) were shown to be vulnerable to attacks. There are a variety of reasons security protocols have flaws. Of course, the first thing to notice is that protocol security is an undecidable problem [118]. Hence one can argue about the security of a given protocol in terms of it's vulnerability to a range of attacks, but cannot guarantee that a protocol is secure against any given attack, using any given technique, at any time. Security protocols involve the exchange of messages between parties with the intent of establishing a level of trust between the parties. This level of trust may involve activities such as one-way or mutual authentication of identities, or secure establishment of an encryption key to be used for the current session (a session key). The number, format and content of messages exchanged varies with the individual protocols. The risks to these protocols are dependent upon the actual goals of the protocols. For example, a one-way authentication protocol is not at risk of a failure of mutual authentication, since that is not a goal. Regardless of the goals of the protocols, researchers and practitioners have found many published protocols to be flawed in that the participants complete the protocol assuming the goals have been met when in fact they haven't. These faults occur when an adversary uses specific properties of the protocol to fool it into accepting messages that it shouldn't. To illustrate the kind of attacks to which security protocols can be vulnerable, we present a number of well-known strategies that a penetrator can employ. However, please note that this list is not exhaustive but can serve to illustrate various styles of attack. Also, these attacks that we present here are only the ones due to a flaw in the protocol design. There can be other types of attacks such as, cryptanalytic, monitoring timing, EM radiation in power consumption etc. These attacks are not due to a flaw in the protocol but in the underlying cryptosystem.

1.5.1 Reflection

In reflection attacks [234], the trick is to fool an agent by sending a message created by the agent, back to himself. An example is to bounce messages back at an agent so that he reveals the password. Sometimes this can happen. A simple analogy might be responding to a guard's question, 'Tell me the password?' with 'Tell me the password?' to which, if he is programmed to respond automatically to that request, he might respond with the password. Then, the same password can be supplied back to him. This is a simple attack but in fact, such an attack has been used against some real friend-or-foe type protocols. Observe that the attack depends on the symmetry of the situation. In the above example, we have assumed that the guard was programmed to authenticate himself and using the same password as people are approaching. Breaking the symmetry, for example, by using a different password than the one which people approaching him use to authenticate themselves, would foil such an attack.

1.5.2 Man-in-middle

A *man-in-the-middle* attack, as the name suggests, works when a penetrator sits in between two communicating agents [234]. Usually in these attacks, the penetrator has complete control over the network and can read all the traffic, obstruct messages from one participant from reaching the other participant and can create his own messages and send them to other agents. Let us look at the following naïve protocol to illustrate one such attack:

Let Alice and Bob be the agents wanting to communicate secretly (playing the roles of A and B respectively). Assume that both do not know each others private and public keys. Let Yves (denoted as Y) be the intruder. Also let PK_A denote the public key of Alice.

1. $A \rightarrow B : \{X\}_{PK_A}$

   Alice sends Bob, an element X encrypted with her public key. Therefore, this can be only be decrypted by Alice's private key which only Alice has.

2. $B \rightarrow A : \{\{X\}_{PK_A}\}_{PK_B}$

   When Bob receives this message, he cannot decrypt the message. Only Alice can do this. But, Bob can encrypt this further with his public key. He sends this back to Alice.

3. Now, using the commutative property of RSA [220], we have

   $$\{\{X\}_{PK_A}\}_{PK_B} = \{\{X\}_{PK_B}\}_{PK_A}$$

   Therefore, Alice can strip off her encryption and give, $\{X\}_{PK_B}$.

4. Alice can send this back to Bob which he and only he can decrypt.

This protocol seems pretty secure at first glance. But it turns out that intruder Yves can easily subvert the protocol by intercepting the messages between Alice and Bob and inserting some of his own. The attack can be shown as below:

1. Yves intercepts the first message from Alice and encrypts with his own public-key.

$$\{\{X\}_{PK_A}\}_{PK_Y}$$

2. This he returns to Alice and Alice has no way of knowing that this is not the one she expects from Bob. So, she strips off her encryption according to the protocol and sends back to Bob:

$$\{X\}_{PK_Y}$$

3. Yves again intercepts this before it can reach Bob and decrypts with his private key to know X.

This attack is successful because of the lack of any form of authentication in this protocol. Alice has no way of checking that the message that she gets back has indeed come from Bob.

1.5.3 Oracle

Here the intruder tricks an honest agent into inadvertently revealing some information that later helps the intruder to attack the protocol [234]. Common examples of these attacks are ones which also use some other technique such as interleaving to attack the protocol. These attacks might also involve multiple runs of the protocol. In fact, they can involve the use of information from runs of entirely different protocols. We shall present many examples for all these attacks including explanations as to how they occur and the mechanism to prevent these attacks in the sections to follow.

1.5.4 Interleaving attacks

We now present some attacks which have been by far the most popular among attacks on security protocols. They are called interleaving attacks. As the name suggests, they work because of interleaved runs of protocols. We shall illustrate this attack with a famous attack that combines interleaving with oracle techniques. The protocol we consider is the Needham-Schroeder Public-Key protocol we introduced in section 1.4

$$\begin{array}{ll} \text{Msg 1.} & a \rightarrow b : \{a, n_a\}_{PK_b} \\ \text{Msg 2.} & b \rightarrow a : \{n_a, n_b\}_{PK_a} \\ \text{Msg 3.} & a \rightarrow b : \{n_b\}_{PK_b}. \end{array}$$

Another examination of the protocol would suggest that, after finishing the protocol run, Alice and Bob might feel confident that:

- they know with whom they have been communicating;

- they agree on the values of the nonces n_a and n_b;
- no one else knows the values n_a and n_b.

These beliefs seem reasonable given the fact that only Bob can decrypt a message encrypted with his public key and similarly for Alice. Alice and Bob can use these values for re-authentication at a later time or even perhaps use the hash of these values as a shared session key for later secret communication. It was believed that this protocol satisfies exactly these properties for many years, until a very neat vulnerability on it was discovered by Lowe [162]. The attack proceeds as follows:

Msg α.1 $A \to Y : \{A, N_A\}_{PK_Y}$
Msg β.1 $Y(A) \to B : \{A, N_A\}_{PK_B}$
Msg β.2 $B \to Y(A) : \{N_A, N_B\}_{PK_A}$
Msg α.2 $Y \to A : \{N_A, N_B\}_{PK_A}$
Msg α.3 $A \to Y : \{N_B\}_{PK_Y}$
Msg β.3 $Y(A) \to B : \{N_B\}_{PK_B}$.

In this attack, Yves is actually a recognized user. He has his own certified public key and private key and is known to the other users. Alice innocently starts the protocol with Yves and sends the first message. Yves takes this message and uses it as the first message in a communication using the same protocol with Bob. Bob also sends the second message, nonces N_A and N_B concatenated and encrypted with Alice's public key. Yves now takes this encrypted chunk and sends it to Alice as the second message of it's run with Alice. Alice then sends Yves the nonce N_B encrypted with Yves' public key as a part of the third message in the run α. Yves can read this value since he can decrypt it with his private key. He then constructs the final message of the run with Bob, using this nonce and Bob's public key. So, at the end of this we have two interleaved runs of the protocol with Yves communicating with Alice and Bob concurrently. Alice thinks Yves and she exclusively share the knowledge of N_A and N_B. Bob thinks he is communicating with Alice when in fact, he is communicating with Yves. Thus, the attack has created a mismatch in Alice's and Bob's perception, at the very least.

This vulnerability has evoked a lot of interest and controversy in the community. The protocol was first published in the seminal paper by Needham and Schroeder in 1978 [201]. The protocol was subjected to analysis using the BAN logic of authentication and was given a clean bill of health. The protocol was thus thought to provide a solid mechanism for authentication until the above attack was published. This attack slipped through the BAN analysis since it falls outside the assumptions made by the BAN logic (BAN explicitly assumes that all the users are honest).

The attack is a combination of interleaving and oracle. Two runs have been successfully completed, namely, α and β. In steps 2 and 3 Yves is using Alice as an oracle to decrypt the message from Bob that he himself cannot decrypt. She is thus fooled in one run into providing Yves with information that Yves can use to successfully run an attack in another run with Bob.

1.5.5 Type Flaw Attacks

Heather *et al.* [130] (page no 1) define type flaw attacks as:

"A type flaw attack on a security protocol is an attack where a field that was originally intended to have one type is subsequently interpreted as having another type."

Example:

Consider the seven message version of the adapted Needham-Schroeder Public-Key Protocol [162] in the section 1.3. We produce it again to illustrate a type flaw attack:

Msg 1. $a \rightarrow s : b$
Msg 2. $s \rightarrow a : \{PK(b), b\}_{SK(s)}$
Msg 3. $a \rightarrow b : \{n_a, a\}_{PK(b)}$
Msg 4. $b \rightarrow s : a$
Msg 5. $s \rightarrow b : \{PK(a), a\}_{SK(s)}$
Msg 6. $b \rightarrow a : \{n_a, n_b, b\}_{PK(a)}$
Msg 7. $a \rightarrow b : \{n_b\}_{PK(b)}$.

Meadows [187], describes a type flaw attack on the unmodified Needham-Schroeder Public-Key Protocol. The modified version of the protocol is also vulnerable to essentially the same attack [130]:

Msg α.3. $I_A \rightarrow B : \{N_I, A\}_{PK(B)}$
Msg α.4. $B \rightarrow S : A$
Msg α.5. $S \rightarrow B : \{PK(A), A\}_{SK(S)}$
Msg α.6. $B \rightarrow I_A : \{N_I, N_B, B\}_{PK(A)}$
Msg β.3. $I_{(N_B,B)} \rightarrow A : \{N_I, (N_B, B)\}_{PK(A)}$
Msg β.4. $A \rightarrow I_S : (N_B, B)$
Msg α.7. $I_A \rightarrow B : \{N_B\}_{PK(B)}$.

Two runs are used in this attack, the messages of which are labelled α and β; I_A denotes the intruder I faking a message, apparently from A, or intercepting a message intended for A. The intruder aims to impersonate A throughout run α. When B sends the nonce challenge at message α.6, the intruder replays this message at A, as message β.3; A interprets the field (N_B, B) as being an agent's identity, and so believes this message came from (N_B, B). So, A tries to request (N_B, B)'s public key, by sending the 'participant identity' (N_B, B) to the server; this allows the penetrator to learn N_B, and hence respond to the nonce challenge. Type flaw attacks on some other protocols were also presented including the Woo and Lam protocol that we presented in the previous section. Heather *et. Al* [130] propose a solution to prevent type flaw attacks on security protocols using a technique of tagging each field with some information indicating its intended type. They also prove their claim using the strand space framework presented by Thayer *et al.* [102].

1.5.6 Multi-Protocol

Alves-Foss [27] defines a multi-protocol attack as

"an attack against an authentication protocol that uses messages generated from a separate protocol (not just another run of the same protocol) to spoof one of the participants into successfully completing the protocol."

Example

Consider the protocol below:

Msg i. $B \rightarrow A : B, A, \{M, N_b, B\}_{PK(A)}$
Msg ii. $A \rightarrow B : A, B, \{N_b, B\}_{PKS(A)}$

Protocol 1.

Alves-Foss [27] describes a possible attack when this is combined with the Needham-Schroeder-Lowe protocol (modified NS):

Modified NS protocol **Tailored Protocol**

Msg 3. $E_A \rightarrow B : A, B, \{N_a, A\}_{PK(B)}$
Msg 6. $B \rightarrow E_A : B, A, \{N_a, N_b, B\}_{PK(A)}$

Msg i. $E_B \rightarrow A : B, A, \{M = N_a, N_b, B\}_{PK(A)}$
Msg ii. $A \rightarrow E_B : A, B, \{N_b, B\}_{PKS(A)}$

Msg 7. $E_A \rightarrow B : A, B, \{N_b\}_{PK(B)}$

Multi-Protocol Attack against Modified NS

1. Intruder E_A, masquerading as A, sends message 3 to B. The tailored protocol requires N_a and M to be in the same format.
2. The intruder will then receive a response from B in message 6.
3. The intruder then forwards this message as message i for the tailored protocol to user A.
4. A responds with message ii of the tailored protocol, that includes a publicly readable copy of the nonce N_b.
5. The intruder grabs this value to create message 7. Then he sends it to B who then validates the intruder's identity as A.

The attack works because of the ability of the intruder to force A to decode the secret field N_b from message 6 and sent it back to the intruder in a format that the intruder can read. Since the format of message i of the tailored protocol is exactly the same as message 6, there is no way that A can detect the attack.

It is interesting to ask if every protocol vulnerable to such "tailored protocols." Indeed, the concept of tailored protocols is trivial—any protocol can be attacked if the simultaneous operation of such tailored protocols exist. Kelsey *et al.* [140] go on to prove that given a protocol there always exists another protocol such that the original protocol can be attacked.

This concept of tailored protocol is only for illustration purposes. In practice it is improbable that honest users are convinced to install and use those tailored protocols to attack the original protocol. However, reality is often a case of mixed protocols, if not tailored protocols. For example, protocols like Kerberos and Neuman-Stubblebine [147, 206] have re-authentication parts which have the same message formats as the primary protocol. Also, protocols like IKE, ISAKMP [126, 180] etc. have different numbers of options and hence different protocols. In most occasions the same user is involved in all the protocols.

After much discussion and debate for years on multi-protocol attacks, Guttman *et al.* proved a useful result that the security of one protocol can never be undermined in the presence of another protocol if both use disjoint encryption. We will explain more about this in a later section.

1.5.7 Replay

A *replay attack* on a security protocol is an attack involving the replaying of messages from a different context than the intended context, thereby fooling an honest agent into believing that the message did originate from the right participant. There have been many definitions of replay attacks in literature. It is a term that tends to be loosely defined and is often interpreted in different ways. However, we can classify them broadly as attacks on security protocols wherein:

1. A message (or parts of message) is replayed from another run of the same or different protocol (external attacks)

   –*OR*–

2. A message (or parts of message) is replayed from the same run (internal attacks).

Multi-protocol attacks and type flaw attacks that we presented earlier are a subset of replay attacks. While multi-protocol attacks involve replaying or using messages from a different protocol, type flaw attacks involve replay of messages by the intruder, that were created by the honest participant, due to lack of knowledge about the types of fields in the messages. These may be outside or inside the current protocol run.

The term replay attacks covers a much broader class of attacks that encompass interleaving, multi-protocol and type flaw attacks. They cover the cases wherein messages are replayed from:

1. A different run of a different protocol;

2. A different run of the same protocol; or,

3. Inside the same run of the protocol.

Other replays also exist, such as straight replays, reflections etc. which are a variation of the above replays. (Reflections have been already covered in the section on reflection attacks. Deflections also exist which are just a variation of reflections in the sense that they involve deflecting the message to a third party instead of reflecting them to the sender himself, as in reflection attacks). Syverson [252] gives a taxonomy of replay attacks. He classifies replay attacks on cryptographic protocols in terms of message origin and destination. The taxonomy is independent of any method used to analyze or prevent such attacks. He claims that it is also trivially complete in the sense that any replay attack is composed entirely of elements classified by the taxonomy. The taxonomy is more than a list of replay types. This means that the classification is hierarchical and each level in the hierarchy forms a partition at the preceding level. All replays can be classified as falling into one of the categories at each level of the hierarchy. The taxonomy is broadly classified into:

1. Run external attacks (replay of messages from out side the current run of the protocol) and
2. Run internal attacks (replay of messages from inside the current run of the protocol), similar to the classification that we presented.

Let us illustrate replay attacks using a classic example, the replay attack on the BAN-Yahalom Protocol [72] published in [253]:

Msg 1. $A \rightarrow B : A, N_a$
Msg 2. $B \rightarrow S : B, N_b, \{A, N_a\}_{K_{bs}}$
Msg 3. $S \rightarrow A : N_b, \{B, K_{ab}, N_a\}_{K_{as}}, \{A, K_{ab}, N_b\}_{K_{bs}}$
Msg 4. $A \rightarrow B : \{A, K_{ab}, N_b\}_{K_{bs}}, \{N_b\}_{K_{ab}}$.

As before, let Alice, Bob and Server play the roles of A, B and S. Alice is the initiator in this protocol. She sends her own name and a random number that she will use to verify the freshness of later messages that contain it (using a nonce) to B. Next, Bob sends to the server his own nonce, and the previous message that Alice has sent, encrypted with the secret key that he shares with the server. In the third message, the server sends to Alice, a message that tells Alice that the server has been talking to Bob, that the message is fresh (through N_a), gives her the session key K_{ab} and an encrypted chunk for her to pass on to Bob. The first encrypted part, ($\{B, K_{ab}, N_a\}_{K_{as}}$), tells Alice that the server has been talking to Bob. The second encrypted chunk ($\{A, K_{ab}, N_b\}_{K_{bs}}$) is for her to pass on to Bob. When Alice sends this to Bob in the fourth message, it tells him that the server has recently talked to Alice and gives him the session key. Because Alice has used the session key to encrypt Bob's nonce, the second encrypted chunk lets him know that she has seen the session key recently. Now, the attack works as follows:

Msg 1. $A \rightarrow B : A, N_a$
Msg 2. $B \rightarrow S : B, N_b, \{A, N_a\}_{K_{bs}}$

Msg 1′. $Y_a \rightarrow B : A, (N_a, N_b)$
Msg 2′. $B \rightarrow Y_s : B, N_b', \{A, N_a, N_b\}_{K_{bs}}$

Msg 3. *Omitted.*
Msg 4. $Y_a \rightarrow B : \{A, N_a(= K_{ab}), N_b\}_{K_{bs}}, \{N_b\}_{K_{ab}}$.

Here, Y_X refers to the penetrator Yves, masquerading as principal X. It is assumed that the message is intercepted if this occurs in the place of an intended recipient.

This attack begins with the penetrator either eavesdropping on an initial message from Alice to Bob or sending it herself. Yves initiates another run of the protocol at the second message, masquerading as Alice. He uses nonce N_a concatenated with N_b (both sent by Alice and Bob respectively in the first run) as nonce N'_a in the second run. He intercepts the encrypted message sent by Bob to Alice in message $2'$ and drops the second run. He then uses this encrypted portion as part of the fourth message of the first run. And so, he uses nonce N_a as the shared key K_{ab} and sends it to Bob. He can now encrypt N_b with K_{ab} since he now 'knows' K_{ab} (nonce N_a). At the end of the protocol run, the intruder is able to make Bob believe that N_a is the secret key. In other words, Bob would communicate with Yves thinking that he is Alice and send him information using the secret key $K_{ab}(= N_a)$.

Although this attack is published as a replay attack, it falls into the sub-category of type flaw attacks that we presented earlier. The attack assumes that substituting two concatenated nonces for one will go undetected and be passed along when sent to someone who has no need to check the nonce. It also assumes that substituting a random number generated to be a nonce for something that has been generated as a session key will be successful. We will later explore replay attacks further, since this thesis is centered around preventing them.

Many solutions have also been presented as countermeasures for replay attacks [55, 92, 123]. The existence of interleaving attacks has prompted occasional discussion of the inappropriateness of freshness mechanisms for general prevention of replay. Some have proposed in their solutions a mechanism to tie the messages to a particular protocol run rather than to a particular epoch. (Timestamps for example were proposed for tying messages belonging to a particular epoch). So, if a replay attack in the form of interleaving involves fresh messages, it would still be revealed by the mechanism if the messages are from different protocol runs. However, the possibility of run internal attacks shows that a mechanism is incomplete if it is not a general scheme to prevent replay attacks. Also, there is the case of reflections and deflections. Some examples of countermeasures were discussed in [194, 117]. They involve indicating who a message is from, who it is to, or both. They propose asymmetry between messages X sends to Y and those Y sends to X as a simple means of countering replay. This is also proposed in [67]. However, these mechanisms should be only expected to be effective in case of reflections. To introduce effective asymmetry in a protocol format, the problem of preventing type flaws must be addressed. A very important step in making sure that format asymmetry is not itself attackable was made by Heather *et al.* [130], as mentioned earlier. So, the proposed solutions using asymmetry can be expected to be effective. Our proposed solution that we present here can be considered as a mixture of all these schemes. Our solution is to indicate in each encrypted component in a protocol run an identifier for the run, it's intended sender and recipient, and the component number of the component in that protocol. This information should be attached to the contents inside every encrypted component in the form of a tag so that when a participant receives an encrypted component, he knows the origin, run and the intended place of the component in the protocol. When a penetrator tries to fool an honest agent by replaying an encrypted component from a different context, he cannot change the tag placed inside it when it was created, and so will be caught when the recipient reads the tag. It is not necessary that the recipient be able to decrypt a component when he receives it. Many times, guided by the protocol rules, he might only be able to decrypt it a later stage of the protocol. However, no matter when he decrypts it, if he notices that the tag indicates that the message belongs to another context, he would stop the protocol run.

1.6 Analysis Approaches

Security protocols require careful analysis. This was recognized early on. It was also realized that just poking around a protocol until one gets got bored and fails to locate an attack does not provide any good assurance that the design is sound. We shall give an overview of some rigorous analysis approaches in this section. These approaches have been proposed to make the reasoning about security protocols more systematic, formal and in some cases, automated. However, this is not an exhaustive survey; but enough information is provided so that a reader, if interested in pursuing further studies on this analysis, can do so using this introduction. The area has seen an explosive growth, with a number of formalisms being used for analysis. Broadly speaking they fall into four main categories:

- logic-based
- model-checking, state enumeration
- proof-based
- cryptographic (provable security).

Recent trend has been towards combining these. For example, bringing together model-checking and proof-based techniques and tools looks to be a very fruitful way to go for areas other than security protocols, such as critical systems in general.

1.6.1 Dolev-Yao Model

This was probably one of the significant steps in the development of the subject. Dolev and Yao, in their paper [97], laid the conceptual foundations of the subject by presenting the basic intruder model that has been used in virtually all the work since. The idea of an intruder with all the capabilities of manipulating messages, passing over the system, flushing, replaying, faking, redirecting and so on, limited only by cryptographic constraints and knowledge of keys was set out. The paper has also given an introduction to the concept of viewing the problem as a form of word problem.

1.6.2 BAN Logic

Burrows, Abadi and Needham [72] made one of the first attempts to make the reasoning about the properties of security protocols more systematic. They proposed a logic called BAN logic of authentication. The idea is to reason at each state the beliefs of agents (legitimate) involved. i.e., deriving the beliefs of such agents as new information is received. For this, the initial knowledge, assumptions and the protocol steps are mapped into formulae in the logic. This process is known as *idealization*. Many flaws in proofs using BAN logic have resulted from faulty idealization. BAN logic is really about authentication. This is a fact that is often overlooked, although the authors clearly describe it in their formulation. Another important point is that BAN explicitly assumes that all participants are honest. This is of course not a universally applicable assumption, but a valid one in some contexts. In a distributed network environment like the internet, where assuming that no malicious netizens would be involved in the system, can be so misleading that applying this logic would lead to many erroneous results. If BAN is applied to contexts outside it's assumptions, the results can be seriously misleading as exemplified by Lowe attack on Needham-Schroeder Public-Key Protocol [162].

Let us look at some of the formulae involved in BAN logic. These are not exhaustive but would give a fair idea as to how to use the formalism. First,

$$P \mid\equiv X.$$

This is pronounced, 'P believes X'. More precisely, it should be interpreted as 'P has a good reason to believe X'. Next we have

$$P \xleftrightarrow{\mathrm{K}} Q$$

which is interpreted as: 'the key K is good for use in communication between P and Q' . This should be interpreted as meaning that K is only and will only be known to P and/or Q (of course assuming that P and Q themselves do not compromise K). Third, there is

$$\sharp(X).$$

It is pronounced, 'the term X is fresh' . This notion has been hotly debated. This does not state how old the value is. BAN distinguishes only the present and past, a weak notion of time. Present means from the start of the current protocol run under consideration. Past means everything before the current run. Some pitfalls can be easily observed here. In the asynchronous universe, having a global starting time at which the protocol starts may not be well-defined. Anyway, $\sharp(X)$ is really asserting, 'X has not been sent in a message before the current protocol run' . Next there is

$$P \triangleleft X$$

which is read, 'P sees X' . In other words, 'P receives X'. X might be a term inside a compound term and might require decryption by P. In that case, it is assumed that P would have the appropriate key. Finally, we have

$$P \mid\sim X$$

This means 'P once said X', i.e., P has in the past sent a message that had X in it and, also, P believed X at the time of sending this message.

Examples of the inference rules include:

$$\frac{\begin{array}{c} P \mid\equiv (P \xleftrightarrow{\mathrm{K}} Q) \\ P \triangleleft \{X\}_K \end{array}}{P \mid\equiv (Q \mid\sim X).}$$

This should be interpreted as saying: 'if P believes that the key K is good for communication between it and Q, and also if P has seen X encrypted with the K, then it can be inferred that P now believes that Q has once sent X'. (The logic assumes that principals can identify their own messages). There are more rules that deal with the notion of jurisdiction, which concern when agents have the authority to make statements.

Protocol goals like those introduced in section 1.2 (security properties) can also be formulated in the logic. For example, a key establishment protocol would typically aim to achieve as one of its goals:

$$A \mid\equiv (A \overset{\text{K}}{\longleftrightarrow} B)$$
$$\text{and}$$
$$B \mid\equiv (A \overset{\text{K}}{\longleftrightarrow} B)$$

i.e., A and B both believe that the key K is good for communication between them.

The idea is to then see if the protocol goals can be derived using the formulae representing the initial assumptions, protocol steps and the inference rules. Failure to reach the required goals can indicate that the protocol needs to be changed in terms of its details or the need for the addition of further assumptions. On the other hand, the analysis can sometimes identify places where the assumptions are unnecessarily strong or the protocol can be further simplified, for example an unnecessary encryption of a term. The BAN logic has proved itself highly effective in this sense. But, for high-assurance applications, the precise interpretation of a successful BAN analysis is not very clear. Questions like, 'can a principal believe something that is false?' crop up. This was the problem with the original semantics. They were found to be problematic and raised such questions. Subsequently, an improvement in the semantics was made but it still remains a fact that it is quite difficult to interpret the results of a BAN analysis. This is true specifically because the protocol steps must be idealized into the logic a step which may accidentally introduce implicit assumptions.

1.6.3 NRL Analyzer

The NRL analyzer [187, 186, 188] falls in between a model-checker and a theorem prover. Model checkers involve specifying all the properties of a protocol including rules, assumptions, intruder capabilities and protocol states in a generic model. A general procedure is followed to analyze the protocol to find flaws or attacks. Theorem proving involves verifying the correctness of theorems using automated tool support.

The NRL analyzer has been around for some time. It is still going strong. It breaks away from traditional analysis approaches and specifies the problem as a word problem. It is a prototype special-purpose verification tool, written in Prolog. It uses one of the classical concepts of artificial intelligence called back-tracking in the analysis of cryptographic protocols.

NRL is based upon a version of the term-rewriting model of Dolev and Yao [97] mentioned earlier. In the Dolev-Yao model, cryptographic protocols are thought of as an algebraic system that obey a set of reduction rules. For example, encryption and decryption with the same key using a private-key algorithm is self-cancelling. Thus, they model the intruder as attempting to solve a word problem in a term-rewriting system. Using this insight Dolev and Yao, and later

Dolev, Even and Karp [96], developed a set of algorithms for proving the security of certain limited classes of protocols. NRL is based on this model but takes a more general approach. For one thing, it extends the goals of the intruder to include more than just finding out secret words. Also, the model is extended to include local state variables possessed by the principals. This is because, for example, a protocol can be broken if the intruder can convince a principal that a word already known by the intruder is a session key. This models the scenario of protocols being broken, not by the intruder discovering a secret word, but by the intruder convincing a principal that a word has certain properties that it does not have.

While Dolev and Yao give a set of algorithms, NRL was developed based on a general procedure for proving security properties of protocols and an interactive Prolog program that facilitates this procedure. Protocols are specified as a set of transitions in a state machine. According to Meadows [186], (page no. 4), each transition rule is specified in terms of:

1. words that must be input by the intruder before a rule can fire;
2. values that must be held by local state variables before the rule can fire;
3. words output by the principal (and hence learned by the intruder) after the rule fires; and,
4. new values taken on by local state variables after the rule fires.

In essence then, the NRL Analyzer is an equational re-writing tool, written in Prolog. It also incorporates automated support to assist the user in proving certain impeachability theorems that serve to prune the search space by helping to discard infinite nodes of potential state space. The tool is run as a backward search from a specified insecure state to see if the state could be reached from the initial state. The search is partly automatic. Sometimes the tool requires the user to intervene. Typically, when the search space is growing explosively, the user needs to intervene and prove some more lemmas that aid in cutting the branches that are leading the tool to go into an infinite loop.

Using the tool requires quite a high level of user expertise. Protocol rules have to be accurately coded and the insecure states from which the search is to be driven have to be identified. The search typically needs a high level of expert user interaction (proving impeachability theorems and lemmas etc. to prune the search space). The tool has been used to great effect to analyze a number of flaws in a number of protocols. In some cases, new flaws have been discovered on famous protocols like the Simmons broadcast protocol [242].

1.6.4 Strand spaces

The strand spaces approach has been developed by Thayer *et al.* at MITRE [102, 101]. A strand represents a sequence of communications for an agent involved in communication. For an honest principal, this encodes the expected sequence of send and receive messages associated with a particular role of the protocol. Agents can play multiple roles in a protocol (sometimes simultaneously). For example, the role played by 'a' in the Needham-Schroeder-Lowe Public-Key protocol can be represented by the following strand:

$$\langle +\{N_a.a\}_{PK_b}, -\{N_a.N_b.b\}_{PK_b}\rangle.$$

'+' stands for *send* and '$-$' for *received.* Similarly, the responder strand (agent 'b') is written as

$$\langle \{-N_a.a\}_{PK_b} \rangle.$$

Formally, it is a sequence of the form $\langle \pm f1, \pm f2, \ldots, \pm fn \rangle$, where $+f$ represents the transmission of a message f and $-f$ represents reception of f. A *node* is any particular communication $\pm f$ by a single agent.

A graph structure is defined on strands by means of two types of edges denoted by $\rightarrow$ and $\Rightarrow$: A *protocol* is represented in terms of strands of roles in the protocol. A *bundle* is defined as a collection of strands representing a particular history of the network.

Penetrator strands are also defined in terms of the possible actions by the penetrator. These are parameterized by K_P as the set of keys initially known to him. These strands include sending text messages, concatenating two messages, separating two messages, sending a key, encrypting a message after receiving an encryption key and decrypting an encrypted message to send the plain-text message upon receiving the corresponding decryption key.

Reasoning takes place on sets of bundles. One can imagine this as giving the set of all possible words consistent with a particular agent's experience of a protocol run. With this framework in place one can prove various security properties of protocols. The strand space framework was originally proposed to prove security protocols correct, and so they were not initially used for any other purpose, particularly the other direction in security protocol research—finding flaws. Usually when using the strand space framework the protocol model is first stated. Then, some useful lemmata are proved which have general utility in proofs. Since bundles form finite, well-founded sets under the causal (partial) ordering, standard proof techniques for such structures can be applied. Usually, induction, transformation and general properties of set theory are applied.

1.6.5 Provable security

All the above frameworks we have discussed have been the ones that have drifted away from the details of the underlying cryptosystems, cryptographic algorithms and primitives. There also some well-known ways to characterize the security of cryptographic primitives. These involve theories of probability and complexity. They are quite technical and require in depth understanding to interpret their exact mechanisms. Details of some of those techniques can be found in [64] and [213]. They typically boil down to a sort of process equivalence or at least approximate equivalence. Firstly, the intruder is modelled as being able to perform an unbounded number of tests. One example is asking for the encryption of some piece of text of his choice. He can repeat this procedure possibly basing his choice of input text on the outcome of the previous tests. This procedure reflects what is thought to be the most powerful attack strategies open to the intruder: *adaptive chosen-plaintext attacks.* Finally, he submits two different texts and tries to guess which of the resulting cipher-texts corresponds to which plaintext. A system is considered secure if he cannot do this reliably with significantly better-than-even odds. If he can do it, then the system is deemed insecure. Ideally, the penetrator is deemed to have 'negligible advantage' or for all practical purposes, 'no advantage'. This amounts to placing a tight bound on how much better-than-even odds he can achieve as a function of the amount of work he performs in terms of tests and computation. Again, the details of these are very technical. Numerous attempts have been made to apply this style of reasoning to cryptographic protocols [65, 207]. Almost all of them boil down to reduction-style arguments, e.g. breaking a protocol would be equivalent to breaking the underlying primitive. An ideal thing to do is to combine both styles of analysis: formal and cryptographic. However, a framework that encompasses both aspects of say, the strand space model with the probability and complexity theory of cryptography would almost certainly be intractable. Therefore, it seems better to be able to relate the results of the two styles of analysis [234].

Chapter 2

Background

2.1 Taxonomy of Replay attacks

In this section, we shall present a detailed taxonomy of replay attacks given by Syerson [252]. Our explanation is particularly aimed at describing the various kind of attacks so that a later explanation of the appropriate prevention strategies is easier. The taxonomy is more than a list of replay types. It is a categorization that is hierarchical and each level in the hierarchy forms a partition of the preceding level. Also, we will show that this taxonomy is a complete taxonomy in the sense that all replay attacks can be classified as falling into one of the categories at each level of the hierarchy. The structure is basically determined by whence messages originate and where messages arrive. Penetrator capabilities do not play any role in the classification (other than to affect these two factors). Also, the classification is independent of any ability to detect or prevent penetrator actions. Hence this independence frees us to consider which detection, representation, or prevention mechanism are appropriate for a replay attack by focusing on where it occurs in the taxonomy.

2.1.1 Origination Taxonomy

Origination taxonomy, as the name reflects, is based on the protocol run of origin for a message (relative to the protocol run in which the replay occurs). Origination taxonomy can be classified into:

1. Run external attacks (using replay of messages from outside the current run of the protocol)
   (a) Interleavings (require contemporaneous protocol runs)
   (b) Classic replays (runs need not be contemporaneous)
2. Run internal attacks (using replay of messages from inside the same/current run of the protocol).

There is no distinction between attacks involving replay of parts of a message or the whole message. In fact, intuitively, the significant part of replays is the replay of encrypted components.

We shall not distinguish between messages and parts of messages when considering replays. It is acceptable to use them interchangeably. Also, some attacks may fall into more than one category. i.e., they can involve multiple types of replays to attack the protocol. In that case, it is helpful to think of an attack as consisting of multiple attack elements which may or may not constitute whole attacks. The following is the description of the classes of attacks set out in the above taxonomy.

Run external attacks use message components from one protocol run in another. The protocol run may be using the same protocol or a different protocol (as in multi-protocol attacks described in section 1.4.4). One of the classic examples of this is the Denning-Sacco attack [92] on the original Needham-Schroeder key distribution protocol [201]. The attack requires that a penetrator capture a copy of the third message in the protocol and break the distributed session key. He can then replay this third message later, as part of another run of the protocol (or another protocol using the same message format). By doing so, he can convince the honest participant into thinking he has just authentically exchanged a key with the penetrator. However, the reality is that the key is an old key and he has exchanged it with a penetrator. This forms an external attack because it uses a message from outside of the protocol run in which the replay occurs.

This is an example of a *classic replay*. They are so called because they have been known and addressed for quite some time. They do not need contemporaneous runs[1] of protocol and use a message from outside the current run of the protocol. That message can come from a protocol run that occurred at any time. (The only exception is when the long term keys have expired). The other class of run external attacks are called *interleavings*. These require that two protocol runs overlap in execution. The attack on the BAN-Yahalom protocol presented in section 1.4.6 is an example of this kind of attack. The attack is an interleaving attack because it depends on messages constructed of message components from contemporaneous protocol runs. Observe that if the second run is not begun during the first run, the penetrator cannot successfully complete the attack.

Run internal attacks use message components from within the same or current run of a protocol. An example of this is the attack on the Woo and Lam protocol [272] published by Heather *et. al* [130], where an intruder spoofing as agent a and the server replays the fourth message created by b back to b. Since he has already succeeded in making agent b create an encryption that had a type flaw, he can replay that message as part of another message. Heather *et al.* also presented a technique that prevents this attack in their paper.

2.1.2 Destination Taxonomy

Origination taxonomy is based on the protocol run of message origin relative to the run in which the replay is made. Destination taxonomy focuses on the recipient of the message relative to the intended recipient. There are two kinds of attacks in the destination taxonomy:

1. Deflections (message is deflected to some other participant, other than the intended recipient)
    (a) Reflections (message is sent back to the sender)
    (b) Deflections to a third party

[1]Contemporaneous refers to parallely executing protocol runs.

2. Straight replays (message is delayed).

Deflections involve deflecting the message to other than the intended recipient. These include reflections (explained in section 1.4.1), where the message is sent back to sender and deflecting the message to a third party (i.e., other than the sender and the intended recipient). **Straight replays** are so-called because the message is sent straight from the sender to the intended recipient without obstruction or deflection, although it may be delayed or have additional text appended to it, generally altering the significance of the message.

To illustrate attacks using straight replays, let us look at the following example described in [253]. In this attack, the penetrator reuses a message from Alice to the server (message 2), as a different message from Alice to the server (message 4) in another round of the protocol. The protocol involved in this attack was an artificial example created for illustration of these attacks. However, Syverson also presented these attacks on well-known protocols, like the one below, on the BAN-Yahalom Protocol.

Attack on the BAN-Yahalom Protocol involving straight replay

(1) $A \rightarrow Y_b : A, N_a$

$$
\begin{aligned}
(1') &\; Y_b \rightarrow A : B, N_a \\
(2') &\; A \rightarrow Y_s : A, N'_a, \{B, N_a\}_{K_{as}} \\
(2'') &\; Y_a \rightarrow S : A, N_a, \{B, N_a\}_{K_{as}} \\
(3') &\; S \rightarrow Y_b : N_a, \{A, K_{ab}, N_a\}_{K_{bs}}, \{B, K_{ab}, N_a\}_{K_{as}}.
\end{aligned}
$$

(2) *Omitted.*
(3) $Y_s \rightarrow A : N_i, \{B, K_{ab}, N_a\}_{K_{as}}, \{A, K_{ab}, N_a\}_{K_{bs}}$
(4) $A \rightarrow Y_b : \{A, K_{ab}, N_a\}_{K_{bs}}, \{N_i\}_{K_{ab}}.$

This attack is not as potentially dangerous in terms of the damage that it can cause compared to the attack shown in section 1.4.7. The result is that Yves can spoof Alice into thinking that she has exchanged a key with Bob. The session key is not released anywhere. This attack also does not rely on assumptions of possible type flaws in the messages. The only assumption is that Alice will not detect the reflection in the second run of her nonce from the first run.

This attack also illustrates all of the possible kinds of destination replays. Reflection is involved when Yves, masquerading as Bob, sends message 1 back to Alice in message $1'$. There is also a third party deflection of the encrypted text from message $3'$ in message 3. There is a straight replay of encrypted text from message $2'$ in message $2''$ of the second run.

The origination and destination taxonomies are independent of each other. Therefore, we can combine them to make a full taxonomy of replay attacks (actually it would be a cross product of both. i.e., by appending either one to the finest levels of distinction in the other). However, before we present the full taxonomy, we would like to present two more types of attacks which we consider as 'special' types of replay attacks. They are:

1. Multi-protocol attacks; and,
2. Type Flaw attacks.

Since we have already discussed both these attacks in section 1.4, we shall not discuss them in detail again. But we shall illustrate with examples how both of these fall into the class of replay attacks.

2.1.3 Multi-protocol attacks

As described before, a multi-protocol attack is an attack on a security protocol using messages generated from another protocol to fool one of the participants into successfully completing the protocol. These attacks are possible when the public-key infrastructure permits the use of a user's public key in multiple protocols. An attacker can then use either a different protocol which already exists or a tailored protocol to break an otherwise secure protocol. In other words, these attacks work only when a replay of encrypted components from different protocol runs (that may or may not be using the same protocol) is possible. They are especially successful against a large class of public-key authentication protocols. Particularly, in implementations of the public-key infrastructure where a user's public key can be used for more than one specific protocol, public-key authentication protocols can be broken by launching multi-protocol attacks. Although, theoretically an attack may consist of replays of any text (not just encrypted), an attack on a well-designed protocol can succeed only with replay of encrypted components. Poorly designed protocols (which do not use any form of authentication or encryptions for authentication and key distribution) may be vulnerable even to replays of unencrypted text. Even in well-designed protocols, it might be necessary to replay the unencrypted text along with the encrypted text. However, this is quite trivial. A mere eavesdropping on the network or creation of a similar text can be used to replay unencrypted text. In fact, there cannot exist a mechanism to check for the replay of unencrypted text. Hence, we must acknowledge the fact that the significant and effective part of an attack involves only replay of encrypted text.

A multi-protocol attack involves a similar replay of encrypted text from another protocol (not just the another run of the same protocol) in the current protocol. Hence, the techniques to prevent these attacks consist of identifying uniquely each encrypted text (with respect to the protocol in which they are/were used). We shall discuss more about preventing multi-protocol attacks later in the section on strategies to prevent these attacks. Multi-protocol attacks fall into the category of run external attacks since they require at least two protocol runs for the attacks to work.

2.1.4 Type Flaw attacks

Type flaw attacks also involve replay of encrypted text in the wrong context to fool a participant into completing a protocol successfully. Although by definition type flaws mean a flaw in the types of fields in a message, in which case the taxonomy of type flaws is large, the attacks ultimately work only by means of replaying encrypted components that were created using fields having types other than the intended type. Let us illustrate this with an example attack on the Woo and Lam protocol published in [130]:

$$\begin{aligned} &Msg1.\ a \rightarrow b : a \\ &Msg2.\ b \rightarrow a : n_b \\ &Msg3.\ a \rightarrow b : \{a, b, n_b\}_{shared(a,s)} \\ &Msg4.\ b \rightarrow s : \{a, b, \{a, b, n_b\}_{shared(a,s)}\}_{shared(b,s)} \\ &Msg5.\ s \rightarrow b : \{a, b, n_b\}_{shared(b,s)}. \end{aligned}$$

where, $shared(a, s)$ denotes a key shared between a and s. b cannot decrypt the message received in message 3, but simply includes it inside the encryption in message 4. This can be exploited by a penetrator by using a type flaw as follows:

$$\begin{aligned} &Msg1.\ I_A \rightarrow B : A \\ &Msg2.\ B \rightarrow I_A : N_b \\ &Msg3.\ I_A \rightarrow B : N_b \\ &Msg4.\ B \rightarrow I_S : \{A, B, N_b\}_{shared(B,S)} \\ &Msg5.\ I_S \rightarrow B : \{A, B, N_b\}_{shared(B,S)}. \end{aligned}$$

The penetrator replays nonce n_b in message 2 (sent by B) in message 3. Since B has no way of checking the type of the received message (he cannot decrypt the message), he simply includes it inside the encryption in message 4 and sends it to S. Since I_S is masquerading as S, she can simply send this message back to B as message 5. B can now decrypt this message, finds the expected contents and is convinced that he has just finished a successful run with A, when in fact he has run the protocol with I_A.

The attack works because B has created an encrypted component without checking the types of the fields. The penetrator takes advantage of this fact and replays the component as the next message. In general, all type flaw attacks work only when the penetrator replays an encrypted component that was created by an honest participant because of lack of knowledge about it's type.

Type flaws can be used to attack a protocol by replaying messages from inside the current run or from a different run. Hence, type flaw attacks fall into both run external and run internal attacks. Also, they can consist of both deflections and straight replays. The above example attack on the Woo and Lam protocol is an example of a reflection attack occuring in the same run (message 4 from B to I_S is sent back to B in message 5). We now present the taxonomy of replay attacks that include all the above mentioned types of attacks. This is a modified version of the taxonomy given by Syverson [253].

2.1.5 Full Taxonomy

1. Run external attacks (replay of messages from outside the current run of the protocol)
    (a) Interleavings (require contemporaneous protocol runs)
        i. Deflections (message is directed to other than the intended recipient)
            A. Reflections (message is sent back to sender)
            B. Deflections to a third party
        ii. Straight replays (intended principal receives message, but message is delayed)
    (b) Classic replays (runs need not be contemporaneous)
        i. Deflections (message is directed to other than the intended recipient)
            A. Reflections (message is sent back to sender)
            B. Deflections to a third party
        ii. Straight replays (intended principal receives message, but message is delayed)
2. Run internal attacks (using replay of messages inside the same/current protocol run)
    (a) Deflections (message is directed to other than the intended recipient)
        i. Reflections (message is sent back to sender)
        ii. Deflections to a third party.
    (b) Straight replays (intended principal receives message, but message is delayed)

2.2 Strategies against replay attacks

In this section we shall discuss some strategies that were presented in the academic literature to prevent replay attacks. Traditional mechanisms to prevent replay attacks focused on using nonces, timestamps etc. For example, the nonce n_a in NSPK is used to stop a penetrator from replaying b's messages from a previous protocol run into the current run. The nonce is used for 'freshness', or in other words, used for stopping replay of old messages in authentication. However, as we have observed, the use of nonces does not prevent many types of replay attacks, including multi-protocol and type flaw attacks. The existence of interleaving attacks has prompted occasional discussion of the inappropriateness of freshness mechanisms for general prevention of replay. Devices like timestamps which tie messages to a particular epoch were suggested but were also beset with problems such as dealing with the asynchronous universe. Hence, some have suggested mechanisms to tie messages to a particular protocol run rather than a particular epoch. If the interleaving of messages are from different protocol runs, such a mechanism would reveal the replays. Such suggestions have been cited in many places including [73, 55, 123]. However, the question of how to generate such unique protocol run identifiers remains unanswered. Also, these would not stop all kinds of replay attacks, mainly run internal attacks where messages are replayed inside the same protocol run.

As for the destination taxonomy, one kind of countermeasure is one that indicates who a message is from, who it is to, or both. Some examples of these were discussed in [194] to avoid reflection attacks on particular protocols. One of the mechanisms suggested by Mitchell discusses binding the name of the message originator to the message cryptographically. These mechanisms would probably work well against reflections but not against deflections and straight replays. Another mechanism discusses specialized use of shared keys in order preclude mistaking either the sender or the receiver of a message. This will not rule out straight replays and can be very expensive to implement. Some other similar mechanisms suggested by Gong [117] were also explicitly limited to countering reflections.

The basic mechanism is to introduce asymmetry between messages X sends to Y and those Y sends to X as a simple means of countering replay. Again, we should only expect this to be effective in avoiding reflections. Syverson in [253] says, "We must also take care that format asymmetry is not itself attackable." This means that possible type flaws in messages must be avoided to achieve the asymmetry. However, with the help of Heather *et al.*'s work on preventing type flaw attacks on security protocols [130], one can be assured that any given protocol is invulnerable to type flaw attacks and hence be assured that the required format asymmetry will be successfully achieved.

Aura [55] presents some strategies as a set of design principles for avoiding replay attacks in cryptographic protocols. He claims that the principles are easily applied to real protocols and do not consume excessive computing power or communication bandwidth. As opposed to good design principles suggested at many places in literature (which are more like warnings and examples of how protocols should not be built), these are specific instructions for constructing protocols that avoid the pitfalls and satisfy the good principles. These include a claim that the techniques should be implementable in real protocols at a reasonable cost in computation and bandwidth. This is essential because of the tight performance constraints that the designers of concrete protocols often struggle with. Low resource consumption is achieved using inexpensive hash functions and by utilizing the cryptographic functions and redundancy that would in any case exist in protocols. However, as in [23], there is no claim that the ideas are either sufficient or necessary to make all protocols secure. The only claim is that following these guidelines in protocol design results in conceptually simple protocols whose security is easier to reason about. The suggestions are actually listed out in the form of strategies in each section.

1. First consider the concept of implicit typing by using unique functions. Here the suggestions given by Carlsen [73] about the type information that can be attached to messages and data items are cited. In particular, the protocol identifier, transmission step identifier, message subcomponent identifier, primitive type of data items, and protocol run identifier are considered. Except for the protocol run identifier, all the other tags can be represented using static type information by attaching static labels to message data structures. However, full explicit typing is usually avoided for performance reasons. Instead, Aura suggests that creating unique functions for all sub-messages costs significantly less than explicit typing of all messages. So the first strategy is (page no 61),

   > "**Strategy 1.** Use a unique cryptographic function for each sub message in order to tag the sub messages with their static data types. Create the unique functions by parameterizing standard cryptographic functions."

   However, according to Aura, it still remains an open question as to how to tag messages with a *protocol run identifier*. We would call this identifier a *session-id* for a protocol run. One of the aims of this thesis is to answer the generation of such a run identifier through a very specific procedure.

2. Many protocol flaws are caused by the excessive removal of redundancy from the messages in

a protocol. This happens because the protocol designer is tempted to simplify the protocol. But it is often difficult to see which messages are important and which messages are not for the correctness of a protocol. In fact, the more information included in messages, the more difficult it is to commit a replay. Several classic protocol failures would have been prevented by explicitly stating the name of the intended recipient in the encrypted messages [92, 162]. But many times this is too expensive to follow. For example, in mobile communications and smartcard interfaces the use of security technology is severely limited by the high cost of data transfer. Also, in most cases the redundant data is already known by the receiver and is used only for comparison with the expected values. Hence, sending just a conflict-free hash of the data with enough redundancy so that the receiver can reliably compare with the received value would suffice. Instead of full information (known by the sender), the messages can contain:

(a) all information known by the sender but unknown to the receiver;

(b) a cryptographic hash of all information known to both the sender and the receiver.

Aura summarizes this in his second strategy as:

> "**Strategy 2.** Include in all authenticated messages a hash of all information that both the sender and the receiver should agree on at that stage of the protocol run."

Since the run identifier is the one piece of type information that could not be handled with unique cryptographic functions, the idea is to combine unique functions with hashed full information for complete typing of message data.

3. Replay attacks against session data can also arise from confusion about the assumptions behind a key. For example, in a version of the Diffie-Hellman key exchange, where the public keys are stored in public directories, if an attacker copies another person's public key and sends it into the directory as his own key a replay attack can occur. The attacker may not be able to recover the session key but he can replay messages between two sessions because they both have the same session key. The attack can be foiled by requiring the principals to demonstrate the possession of the corresponding private D-H key before accepting a public key to the directory. Hence, the strategy is (page no 66):

   > "**Strategy 3.** Understand the trust assumptions made in the key-agreement protocol and limit the use of the session key accordingly, or change the protocol to eliminate the assumptions."

4. Finally, regarding the use of unique session keys for each session and binding the session key to it's intended use, the following strategies are given:

   > "**Strategy 4.** To prevent copying of session keys and data between sessions with different sets of principals, distribute instead a key derivation parameter and generate the session key by hashing the names of the principals with the parameter";

   > "**Strategy 5.** To bind the session key to it's intended use, distribute instead a key derivation parameter and generate the session key by hashing with the parameter all information related to the key distribution process and the intended uses of the key."

2.2.1 Multi-Protocol Attacks

In section 1.4.4 we discussed multi-protocol attacks. We have also included multi-protocol attacks into our list of replay attacks. We now present the principles and suggestions given in the literature to prevent these attacks.

Firstly it is useful to know the two conditions that enable these attacks:

1. The cryptographic services on the user's machine must permit the use of a public key in more than one protocol. Although it is not impractical to assume that a user may have multiple public keys for multiple levels of security, and possibly for multiple job functions, it has not been seen as necessary in the past for a user to have a separate public key for every security protocol that he/she uses. In fact, it is very expensive to have a separate public key for each purpose. Usually users prefer not to have multiple certified keys because of the cost involved in buying certified keys.
2. The second protocol that enables these attacks needs to be installed on the machine that the masqueraded user is using. Also, the second protocol must have access to the cryptographic services on that machine that utilize the user's public-key or private key algorithms. This protocol need not be a specific tailored protocol but can be an authenticated secure protocol that happens to share message formats with the attacked protocol.

These conditions may exist in a wide range of systems and implementations of the public key infrastructure. To prevent these attacks would mean making either one of these conditions unattainable. Kelsey, Schneier, and Wagner [140] present five design principles for protocols that "appear" to render the chosen protocol attack impossible.

Principle 1. Limiting the scope of the key (to address the first condition above); This needs some mechanism to implement this restriction. Current cryptographic APIs (Application Program Interface) in the market do not impose any restriction on the use of keys other than limiting a key to a particular algorithm and to encryption or decryption. If the application has a handle for the key, it can use it. The APIs have to be designed to manage key use securely.

Principle 2. Identify uniquely each application, protocol, version and protocol step: This is same as condition above. However, identifying uniquely in this manner does not guarantee the security of the protocol. If the tailored protocol matches the attacked protocol in the identifiers, it is still possible to launch an attack. Remember, bad guys lie.

Principle 3. Include a fixed identifier in a fixed place in the cryptographic protocol: However, as Kelsey *et al.* point out, this only prevents the multi-protocol attack using protocols designed with this principle. It does not prevent any attack using *tailored* protocols that do not implement this design suggestion.

Principle 4. Tie the unique identifier (specified in principle 3) to encryption in a way that forces the identifier to be used for successful decryption: This technique prevents a blind signing protocol from being used to decrypt secret messages.

Principle 5. Include support for these mechanisms using smartcards: This is not really a design principle, but rather a statement that infers that a device implementing cryptographic protocols must be able to enforce restriction that render multi-protocol attacks impossible (or at least restrict their scope).

Alves-Foss [27] gives some suggestions to address these design principles. We will not go into those suggestions further. They are just suggestions for implementing these conditions. We meet some of those conditions in this thesis through our approach that we present later.

2.2.2 Type Flaw Attacks

As mentioned earlier, Heather *et al.* [130] define a type flaw attack on a security protocol as an attack where a component that was originally intended to have one type is later interpreted as having another type. Many replay attacks in fact depend on type flaws, including the attack on the BAN-Yahalom protocol that we presented in section 1.4.7. Hence, preventing type flaw attacks prevents large class of replay attacks. Heather *et al.* [130] present a technique that prevents all type flaw attacks on security protocols. In their paper they present a system where fields are tagged with some extra information indicating their intended type. The tag can be thought of as a few bits attached to the field, with different bit patterns allocated to different types, e.g. "(nonce, N)" used to represent a value N tagged in such a way to indicate that it is intended as a nonce. Similarly, compound messages are also tagged; for example, "(pair, ((nonce, N),(nonce, N')))" represents a pair of values N and N' tagged as nonces and so on. Below is an example given in the paper that shows how the tagging prevents type flaw attacks. Consider again the attack on Woo and Lam Protocol presented earlier (section 2.1):

$$\begin{aligned}
&Msg1.\ I_A \rightarrow B : A\\
&Msg2.\ B \rightarrow I_A : N_b\\
&Msg3.\ I_A \rightarrow B : N_b\\
&Msg4.\ B \rightarrow I_S : \{A, B, N_b\}_{shared(B,S)}\\
&Msg5.\ I_S \rightarrow B : \{A, B, N_b\}_{shared(B,S)}.
\end{aligned}$$

If we adopt the tagging scheme, the nonce in message (2) would be $(nonce, N_b)$. The penetrator cannot replay the nonce in this form as part of message 3. However, he can retag the nonce with the tag that B is expecting $(\{|agent, agent, agent, nonce|\}_{shared-key}, N_b)$. Message 4 would then become (without the "pair" tag to simplify the explanation):

$$\begin{aligned}
&(\{|agent, agent, \{|agent, agent, nonce|\}_{shared-key}|\}_{shared-key},\\
&\quad \{(agent, A), (agent, B), (\{|agent, agent, nonce|\}_{shared-key}, N_b)\}_{shared(B,S)}).
\end{aligned}$$

Now this message cannot be replayed as an instance of message 5 because B is expecting a message where the third field inside the encryption is tagged as being a nonce. The intruder can at most change the outer most tag to create:

$$\begin{array}{l}(\{|agent, agent, nonce|\}_{shared-key}, \\ \ \{(agent, A), (agent, B), (\{|agent, agent, nonce|\}_{shared-key}, N_b)\}_{shared(B,S)}\end{array}$$

but he cannot change the inner tag without access to the appropriate key. Hence, the tagging scheme prevents the attack.

Heather *et al.* also present a proof of the efficacy of the tagging scheme in preventing all kinds of type flaw attacks. They also discuss a possible implementation for the tagging scheme. One of the corollaries presented in the paper hints at using a unique component number as part of each encrypted message in a protocol to prevent these attacks. However, there is a subtle flaw in their claim that this component numbering prevents all type flaw attacks. They claim that their original tagging scheme (tagging each part of a message) prevents type flaw attacks involving two different protocols (not merely two different runs of the same protocol). With the component numbering however, this claim is no longer valid. In the presence of a tailored protocol or a secondary protocol which has matching message formats (see attacks in section 1.4.4) as the original protocol, component numbers may not be of any help. For example, two encrypted messages having different formats may have the same component numbers in two different protocols. Therefore when an honest participant sends an encrypted message having a type flaw to an intruder, the intruder can now replay it to another (or the same) participant as part of another encrypted message of another protocol. Since the component numbers now match, even though there is a type flaw, it cannot be detected[2].

This flaw arises because of the inability to uniquely identify each protocol run. Using component numbers identifies each encrypted component in a *single* protocol but not multiple protocols. To identify and prevent attacks involving multiple protocols, we need a mechanism similar to the protocol run identifier suggested by Aura [55] that we presented earlier.

2.2.3 Disjoint Encryption

For a security protocol to achieve a security goal, it does not depend on what can happen but on what cannot happen. As shown in previous sections, when multiple cryptographic protocols are combined the penetrator has new opportunities to obtain the messages which ought to authenticate principals to their peers. Protocol mixing has shown itself to be a significant cause of protocol failure and makes the analysis of protocols more difficult [27, 140, 118]. However, as explained in the section on multi-protocol attacks, in practice it is not uncommon to combine different protocols using cryptography. The purpose of a key distribution protocol, usually, is to deliver a session key to be used for encryption. However, this use may make replay attacks possible if used to construct messages similar to messages used in the key distribution protocol itself. An interesting question is, “Does the use of a key undermine the guarantees provided by the protocol distributing the key?” [123]. In addition to having different protocols running concurrently, protocol mixture is also prevalent. For example, many recent protocols have large numbers of different options, and therefore have large numbers of different sub-protocols inside them [93, 126, 180, 189]. As

[2]The same attack can occur even more readily if we do not use component numbers.

mentioned earlier, if one protocol has encrypted components, all of which are disjoint with the ones used by another protocol (either a tailored or legitimate protocol), in terms of message format, then a replay attack (or a subset like the multi-protocol or type flaw attacks) cannot exist (although, this is an indeed strong assumption). Guttman *et al.* [123] (page no 1) also suggest a rule of thumb for protocol independence when protocols are to be mixed together:

> "Common sense suggests a rule of thumb when protocols are to be mixed together. This rule is that if the primary protocol uses a particular form of encrypted message as a test to authenticate a peer [122], then the secondary protocols should not construct a message of that form. The sets of encrypted messages that the different protocols handle should be disjoint. One way to arrange for this is to give each protocol some distinguishing value, such as a number; that number may then be included as part of each plaintext before encipherment. Then no principal can mistake a value as belonging to the wrong protocol. Another way to achieve disjoint encryption is to ensure that different protocols never use the same key, although this may be expensive or difficult to arrange."

Guttman *et al.* also put forth this principle in the same spirit as those of Abadi and Needham presented as prudent engineering practice for cryptographic protocols [23]. They have proved this concept formally; i.e. they prove that, if two protocols have disjoint encryption, then the first protocol is independent of the second. This means that if the first protocol achieves a security goal (whether a secrecy goal or an authentication goal) when the protocol is executed in isolation, then it still achieves the same security goal when executed in combination with the second protocol. They have used multi-protocol strand spaces [103], to prove that two cryptographic protocols are independent if they use encryption in non-overlapping ways. The idea is to study the penetrator paths, namely, sequences of penetrator actions connecting regular nodes (nodes in the strands of the honest participants) in the two protocols. The concentration is then on showing that the removal of all inbound linking paths (message transmission in the secondary protocol to a message reception in the primary protocol) has no effect if the bundles can be modified and if the encryption does not overlap in the two protocols. The resulting bundle, thus, would not depend on any activity of the secondary protocol. The proof is also exemplified using the Neuman-Stubblebine protocol as an example [103, 206]. Unfortunately, the aproach they suggest requires all installed protocols to follow the these rules or be analyzed for disjointness prior to use.

2.2.4 Our approach

Carlsen presents a list of type information that can be attached to messages and elements [73]. These include protocol identifier, protocol run identifier, primitive types of data items, transmission step identifier and message subcomponent identifier. Aura [55] studies these techniques and comes up with some strategies against replay attacks that are neither necessary nor sufficient but enhance the robustness of a protocol. A recent trend in the literature has been to prove protocol security against specific attacks by tagging messages with one of the type information suggested by Carlsen (eg. [123, 130]).

In the same spirit, we prove that *all* replay attacks can be prevented by tagging each encrypted component with a *session-id* (another name for Carlsen's protocol run identifier) and a *component* *number* (renaming Carlsen's message subcomponent identifier). However, unlike previous attempts, our suggestion is a *general* solution and prevents all types of replay attacks in Syverson's taxonomy. Although, it is not an entirely new solution, it solves the problem of replays using a combined solution that is devoid of any possible vulnerabilities due to interdependencies.

Introducing component numbers inside encryptions is intuitive, but the generation and use of session-ids requires some explanation. Some have discussed tagging messages with all the information that is in possession of a principal and relevant to the protocol [272]. This is also called the *principle of full information.* Aura [55] hints at a trivial way of including a hash of all previous messages in a protocol run, almost as a substitute to the principle of full information and the run identifier. This is prudent to some extent. In fact, as Aura points out, it is enough to include only a hash of the redundant data that is already known to the receiver. This doesn't affect the performance for obvious reasons. However, careful observation of this suggestion reveals a possible vulnerability: Two protocol runs can overlap in the executed information at a typical stage of the runs!

Therefore we suggest a different approach to generating session-ids to identify runs. For the purpose of this discussion, it suffices to know that such identifiers can be used in an effective way and will possess a necessary property—remaining unique to every protocol run. Briefly, all participants need to choose a random number, and combine those into a single long string of random bits. This value should be hashed together with the identities of all principals, reducing the chance of an accidental match in session-ids to a great extent. Two features are necessary to generate such an identifier: 1) Every principal should possess the same hash functions 2) A change in one of the random numbers/principals' identities should make the resulting value differ from it's original value.

Observe that the so-generated session-id is different in properties from other similarly used identifiers. For example, the run identifier used in Otway-Rees protocol [211] is generated by only one participant and hence was shown to be prone to replay attacks [76]. It is also similar to the "cookies" coined in Photuris [139] which are an add-on that can be used to make a protocol more resistant to DOS attacks. A cookie is a unique nonce computed from the names of the sending and receiving parties and local secret information that only the sender possesses. Kent *et al.* in [141] also use cookies, using signed Lamport hash chains [148] to play the roles of cookies, providing weak authentication. Cookies provide initially weak authentication to users while they aid in subsequent establishment of strong authentication. Session-ids are similar to cookies in the kind of initial assumptions and their ultimate use except, unlike cookies where only the sender is aware of the value, a session-id is publicly known.

However, the publicly known identifier needs to be unique for every protocol run. Intuitively, a dishonest principal might use a different value from the pre-agreed upon value. (In fact, this will be definitely true if he replays components from previously completed runs and not from interleaved runs). However, the proof we are going to present will establish that even such behavior does not succeed in breaking a protocol in this scheme. Further, hashing with participants' identities (cited as useful in numerous places including [23, 55, 194, 117]) prevents any other attempts to spoof user identities, and launch man-in-middle type attacks (as similarly used by [141]).

Chapter 3

Strand spaces

The proof that the tagging scheme actually works forms the main part of this part of our work. We prove our claim using a mathematical framework and the concepts and notations from sets, relations and graph theory. In particular, we choose the strand space framework because it is a very convenient framework for the reasoning required in these kind of proofs. The strand space model was originally developed to reason about the correctness of a security protocol. Although it's use is not limited to that, this still continues to be the main use of the strand space model. The approach is distinguished from other works because of the simplicity of the model, it's clean representation, and the ease of developing intelligible and reliable proofs even without automated tool support. In this section, we will outline the concepts in strand spaces using definitions and concepts. Most of the definitions and concepts are taken from [102][1]. We shall also include some lemmas and propositions, the proofs of which can be found in [102]. We exclude proofs in order to explain the model and concepts clearly, without getting too much into the details, thus sparing the reader with the tedious proofs. A *strand* is a sequence of events; it represents the sequence of communications by a legitimate party in a security protocol or a sequence of actions by a penetrator. A *strand space* is a collection of strands, equipped with a graph structure generated by causal interaction. Correctness claims on a protocol may be expressed in terms of the connections between strands of different kinds.

3.1 Introduction

A *strand* is a sequence of events that a single principal may engage in. Each individual strand is a sequence of message transmission and receptions, with specific values of all data such as keys and nonces. Thus, it is a sequential process that exhibits no internal or external choice. For a legitimate principal (honest participant), a strand represents the actions of that principal (of only that party, not it's supposed interlocutor) in one particular run of the protocol. If that particular party may be involved in more than a single run of the protocol during a period of time, each role that the party plays is represented by different strands. The activities of all the parties are represented by their respective strands. A strand for a penetrator is a sequence of message transmissions and receptions that model a basic capability that a penetrator should be assumed to possess. For

[1]In fact, almost all of the definitions have been reproduced with out any change from [102]

example, penetrator strands can include activities such as:

- receiving a key and a message encrypted using that key, and then sending the text inside the encryption;
- receiving two messages, concatenating both and then sending the result;
- sending out a data item such as a name that the penetrator already knows.

Possible penetrator actions may be modeled by connecting a number of penetrator strands.

Strand space models the assumption that some values originate only freshly and uniquely by including only one strand for *originating* that data item by initially sending a message that would contain it. In contrast, many strands may combine with the originating strand by receiving the message and processing its contents further. Also, a strand space models the assumption that some values are impossible for a penetrator to guess - the space simply lacks any penetrator strand in which this value is sent without receiving it first.

3.2 Basic Notions

Let A be the set of elements which are the possible messages that can be exchanged between principals in a protocol. The elements of S are referred to as *terms*. A *subterm* relation is defined on A; $t_0 \sqsubset t_1$ means t_0 is a subterm of t_1. Principals in a protocol can either send or receive terms. The transmission of a term is represented as the occurrence of that term with positive sign, and reception of a term as it's occurrence with a negative sign.

Definition 1. *[102, def. 2.1]*

A signed term is a pair (σ, a) *with* $a \in \mathsf{A}$ *and* σ *one of the symbols* $+, -$. *A signed term is written as* $+t$ *or* t. $(\pm\mathsf{A})^*$ *is the set of finite sequences of signed terms. A typical element of* $(\pm\mathsf{A})^*$ *is denoted by* $\langle\langle\sigma_1, a_1\rangle, \ldots, \langle\sigma_n, a_n\rangle\rangle$.

Signed terms are also treated as ordinary terms, by abuse of language.

Definition 2. *[102, def. 2.2]*

A strand space over A *is a set* Σ *together with a trace mapping* $tr : \Sigma \rightarrow (\pm\mathsf{A})^*$.

A strand space is usually represented as a set of strands Σ. The trace may not be always injective. For example, to model *replay attacks*, we would need to distinguish identical traces occurring at different times.

Definition 3. *[102, def. 2.3]*

Fix a strand space Σ

1. *A node is a pair* $\langle s, i\rangle$, *with* $s \in \Sigma$ *and* i *an integer satisfying* $1 \leq i \leq length(tr(s))$. *The set of nodes is denoted by* $\mathcal{N}$. *A node* $\langle s, i\rangle$ *belongs to the strand* s. *Clearly, every node belongs to a unique strand.*

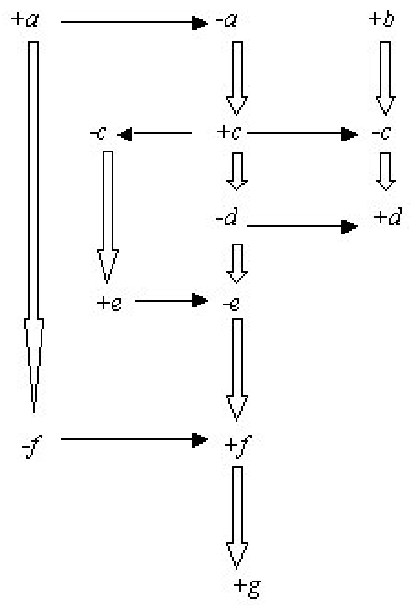

Figure 3.1: A Bundle

2. *If $n = \langle s, i \rangle \in \mathcal{N}$ then index(n) $= i$ and strand(n) $= s$. Define term(n) to be (tr(s))$_i$, i.e. the i^{th} signed term in the trace of s. Similarly, uns_term(n) is ((tr(s))$_i$)$_2$, i.e. the unsigned part of the i^{th} signed term in the trace of s.*

3. *There is an edge $n_1 \rightarrow n_2$ if and only if term(n_1) $= +a$ and term(n_2) $= -a$ for some $a \in \mathsf{A}$. Intuitively, the edge means that node n_1 sends the message a, which is received by n_2, recording a potential causal link between those strands.*

4. *When $n_1 = \langle s, i \rangle$ and $n_2 = \langle s, i+1 \rangle$ are members of $\mathcal{N}$, there is an edge $n_1 \Rightarrow n_2$. Intuitively, the edge expresses that n_1 is an immediate causal predecessor of n_2 on the strand s. $n' \Rightarrow^+ n$ is written to mean that n' precedes n (not necessarily immediately) on the same strand.*

5. *An unsigned term t occurs in $n \in \mathcal{N}$ iff $t \sqsubset$ term(n).*

6. *Suppose I is a set of unsigned terms. The node $n \in \mathcal{N}$ is an entry point for I iff term(n) $= +t$ for some $t \in I$, and whenever $n' \Rightarrow^+ n$, term(n') $\notin I$.*

7. *An unsigned term t originates on $n \in \mathcal{N}$ iff n is an entry point for the set $I = \{t' : t \sqsubset t'\}$.*

8. *An unsigned term t is uniquely originating iff t originates on a unique $n \in \mathcal{N}$.*

A term t can play the role of a nonce or session key in a strand space structure if it originates uniquely in a particular strand space. The set of nodes $\mathcal{N}$, together with both the sets of edges $n_1 \rightarrow n_2$ and $n_1 \Rightarrow n_2$ is a directed graph $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow) \rangle$.

3.3 Bundles and Causal Precedence

A *bundle* is a finite subgraph of the directed graph, $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow)\rangle$. The edges are regarded as expressing the causal dependencies between the nodes. Figure 3.1 illustrates a bundle.

Definition 4. *[102, def. 2.4]*

Suppose $\rightarrow_C \subset \rightarrow$*; suppose* $\Rightarrow_C \subset \Rightarrow$*; and suppose* $C = \langle \mathcal{N}_C, (\rightarrow_C \cup \Rightarrow_C)\rangle$*.* C *is a bundle iff:*

1. C *is finite.*
2. *If* $n_2 \in \mathcal{N}_C$ *and term*(n_2) *is negative, then there is a unique* n_1 *such that* $n_1 \rightarrow_C n_2$*.*
3. *If* $n_2 \in \mathcal{N}_C$ *and* $n_1 \Rightarrow n_2$ *then* $n_1 \Rightarrow_C n_2$*.*
4. C *is acyclic.*

Because C is a graph, it follows from conditions 2 and 3, that, $n_1 \in \mathcal{N}_C$.

Definition 5. *[102, def. 2.6]*

If S *is a set of edges, i.e.* $S \subset \ \rightarrow \cup \Rightarrow$*, then* $\prec_S$ *is the transitive closure of* S*, and* $\preceq_S$ *is the reflexive, transitive closure of* S*.*

Both the relations $\prec_S$ *and* $\preceq_S$ *are subsets of* $\mathcal{N}_S \times \mathcal{N}_S$*, where* $\mathcal{N}_S$ *is the set of nodes incident with any edge in* S*.*

3.4 Terms and Encryption

According to the assumptions ([102], sec. 2.3.),

- A set $\mathsf{T} \subseteq \mathsf{A}$ of atomic messages (texts);
- A set $\mathsf{K} \subseteq \mathsf{A}$ of cryptographic keys disjoint from T, equipped with a unary operator $\mathsf{inv} : \mathsf{K} \rightarrow \mathsf{K}$;

  The assumption is that inv is injective; i.e., it maps each member of a key pair of an asymmetric cryptosystem with the other; and that it maps a symmetric key in a symmetric cryptosystem to itself.
- Two binary operators are defined to produce encrypted messages and concatenating two messages:

$$\mathsf{encr} : \mathsf{K} \times \mathsf{A} \rightarrow \mathsf{A}$$
$$\mathsf{join} : \mathsf{A} \times \mathsf{A} \rightarrow \mathsf{A}$$

It is customary to write $\mathsf{inv}(K)$ as K^{-1}, $\mathsf{encr}(K, m)$ as $\{m\}_K$, and $\mathsf{join}(a, b)$ as $a\ b$. If k is a set of keys, k^{-1} denotes the set of inverses of elements of k.

3.5 Freeness Assumptions

Freeness assumptions are stated in terms of axioms. These axioms basically say that the message terms (set A) are freely generated from the set of texts T and set of keys K using encr and join.

Axiom 1. *[102, Axiom 1] For $m, m' \in \mathsf{A}$ and $K,\ K' \in \mathsf{K}$,*

$$\{m\}_K = \{m'\}_{K'} \Longrightarrow m = m' \wedge K = K'$$

This means that if two cipher texts are equal then the terms inside the encryption and the keys used for the encryption are the same in both the cipher texts. In other words, a different term and/or a different key cannot result in the same cipher text. Usually, this is implemented probabilistically.

Axiom 2. *[102, Axiom 2] For $m_0, m_0', m_1, m_1' \in \mathsf{A}$ and $K, K' \in \mathsf{K}$,*

1. $m_0m_1 = m_0'm_1' \Longrightarrow m_0 = m_0' \wedge m_1 = m_1'$
2. $m_0m_1 \neq \{m_0'\}_{K'}$
3. $m_0m_1 \notin \mathsf{K} \cup \mathsf{T}$
4. $\{m_0\}_K \notin \mathsf{K} \cup \mathsf{T}$.

Each of the conditions are very important in the analysis using the framework. The conditions say that an encrypted text cannot be obtained by concatenating two unencrypted terms, set of terms is *not* closed under the join operation and a key cannot occur inside a term as a subterm. The last condition is particularly important for correctness conditions to hold in many protocols.

Given Axiom 2, width of terms is defined:

Definition 6. *[102, def. 2.10] If $m \in \mathsf{K} \cup \mathsf{T}$ or if $m = \{m_0\}_K$, then $width(m) = 1$. If $m = m_0m_1$, then $width(m) = width(m_0) + width(m_1)$.*

Definition 7. *[102, def. 2.11]*

The subterm relation $\sqsubset$ is defined inductively, as the smallest relation such that:

- $a \sqsubset a$;
- $a \sqsubset \{g\}_K$ *if* $a \sqsubset g$;
- $a \sqsubset g\ h$ *if* $a \sqsubset g$ *or* $a \sqsubset h$.

3.6 The Penetrator

In this section we would show how the penetrator actions can be modeled in terms of his capabilities. His powers are characterized by two ingredients,

1. A set of keys that he already knows (K_P);
2. A set of penetrator strands that allow the penetrator to generate new messages from messages he intercepts.

The atomic actions that the penetrator is capable of are encoded in a set of *penetrator traces.* Following is the list of those atomic actions:

Definition 8. *[102, def. 3.1]*

A penetrator trace is one of the following:

M	***Text message*** $\langle +f \rangle$ *with* $f \in \mathsf{T}$*.*
F	***flushing*** $\langle -f \rangle$*.*
T	***Tee*** $\langle -f, +f, +f \rangle$*.*
C	***Concatenation*** $\langle -f_1, -f_2, +f_1 f_2 \rangle$*.*
S	***Separation*** $\langle -f_1 f_2, +f_1, +f_2 \rangle$*.*
K	***Key*** $\langle +k \rangle$ *with* $k \in K_P$*.*
E	***Encryption*** $\langle -k, -f, +\{(f)\}_k \rangle$*, with* $k \in K_P$*.*
D	***Decryption*** $\langle -k^{-1}, -\{(f)\}_k, +f \rangle, k \in K_P$*.*

This set of penetrator traces guarantees that the penetrator actions are closed under joining, encryption, and the appropriate "inverses".

3.7 Correctness Properties

The original notions of correctness are expressed informally by Thayer *et al.* [102]. For example, they extend Lowe's definition of authentication [164] and state that a protocol guarantees agreement to a participant B (say, as the responder) for certain data items $\vec{x}$ if:

> "each time a principal B completes a run of the protocol as responder using $\vec{x}$, which to B appears to be a run with A, then there is a unique run of the protocol with the principal A as initiator using $\vec{x}$, which to A appears to be a run with B."

A simple notion of secrecy is also stated. A value x is secret in a bundle C if for every node $n \in C$, term$(n) \neq x$. In other words, a value is secret if the regular strands never emit it and the penetrator can never emit it through his possible actions. Honest participants may "know" a secret value in the sense of carrying out computations that depend on it, so long as their behavior in the protocol does not include disclosing it in public. Also, if we can prove that the penetrator never emits a value, it follows that he can never derive it from the values he receives and his available actions since, if he derived it, then he would be capable of emitting it.

We will put these definitions formally as given in [130]. Firstly, we shall define a *template* for a strand.

Definition 9. *Let* Var *be a set of variables such that each variables can be instantiated using a substitution function* sub *defined as a mapping from variables to terms,* $\mathsf{sub} : \mathsf{Var} \rightarrow \mathsf{Term}$ *where* $\mathsf{sub}(v) = f$*,* $f \in \mathsf{Term}$*,* $v \in \mathsf{Var}$*. Then, a strand template is defined as a sequence of signed variables,* (σ, v) *where* σ *is one of* $+, -$ *and* $v \in \mathsf{Var}$*. Also, upon using substitution function* sub *on all the variables of a template, the result should be one of the elements of the trace,* $tr : \Sigma \rightarrow (\pm\mathsf{A})^*$*.*

We can now define correctness properties using *strand templates.* The definitions have been adopted from [130].

3.7.1 Secrecy

The definition of secrecy says that there is a breach of security when there is some strand (of some minimal length) where certain keys have not been compromised, and the value of a particular variable v (intended to remain secret) becomes known to the penetrator.

Definition 10. *[130, def. 1] Let temp be the template for some role; let* $v \in \mathsf{Var}$ *be a variable of temp, intended to remain secret; let* h *be a positive integer; and let* Keys *be a set of keys that the penetrator may not know. A failure of secrecy is said to exist if each of the following holds:*

1. *There is strand* $s = \mathsf{sub}(temp)$ *with C-height at least* h *(i.e. at least the first* h *messages of* s *appear in the bundle* C*).*
2. $\forall k \in \mathsf{Keys} \; \centerdot \; \mathsf{sub}(k) \notin K_P$*.*
3. *There is a node in* C *with label* $+\mathsf{sub}(v)$*.*

The definition says that secrecy is violated if there is a node transmitting a secret value in plain text in the bundle under consideration. If there is a node transmitting the secret value, it means that either the honest agent hasn't sent it or that the penetrator was able to synthesize it using his actions.

3.7.2 Authentication

For authentication, we would consider what it means for a particular role $r2$ to be authenticated to another role $r1$. For authentication to be satisfied, one would expect that whenever there is a strand $s1$ of $r1$, there should be a "corresponding" strand $s2$ of $r2$; these strands should agree upon the identities of the agents involved, and possibly, upon the values of some other variables (e.g. a symmetric key established between them); this is captured by specifying that the strands should agree on the values of all variables from some set X.

Definition 11. *[130, def. 2]*

Let temp1 *and temp*2 *be templates for two roles; let* X *be a set of variables of those templates; let* $h1$ *and* $h2$ *be positive integers; and let* Keys *be a set of keys that may not be available to the penetrator. A failure of authentication is defined as when each of the following holds:*

1. *There is a strand* $s1 = \mathsf{sub}(temp1)$ *with C-height at least* $h1$*.*

2. $\forall k \in \mathsf{Keys}$. $\mathsf{sub1}(k) \notin K_P$.

3. *There is no strand* $s2 = \mathsf{sub2}(temp2)$ *with C-height at least* $h2$ *such that* $\forall x \in X$, $\mathsf{sub1}(x) = \mathsf{sub2}(x)$.

This definition is parameterized by $temp_1$, $temp_2$, X, h_1, h_2 and Keys. $\mathsf{sub1}$ and $\mathsf{sub2}$ are corresponding instantiation functions (def. 9) for roles $r1$ and $r2$ that instantiate the variables in templates $temp1$ and $temp2$ to correspond to elements in T.

Chapter 4

Modelling Protocols

In this chapter, we will present the protocol model used for the proof. But before going further, let us present some assumptions in the model. First, we assume that honest agents will tag messages that they create with the expected tags. For example, if they intend to send a message in a protocol run with session-id, '23456' and in it an encrypted component having a component number '2', they would indeed tag the encrypted component with these two values. On the other hand, we do not assume that the penetrator sends a message with the correct tag. The penetrator is allowed to send a message from any other context and hence with an incorrect tag, reflecting that the message was acquired from some other context, prior to the protocol run or inside the protocol run but under a previous message. We also assume implicit connection determination—all interactions over the network are considered connection oriented. Therefore, participants can be involved in multiple protocols simultaneously, without messages from those protocols being accidentally interleaved (for example, as shown in the attack on the Woo-Lam protocol discussed in [23]).

In the following sections, we will introduce modelling protocols to include tagged facts. Then we will give a brief overview of an adapted strand space model having the newly defined facts. Next, we will introduce strand templates and define what it means for an encrypted fact to be correctly tagged. Lastly, we will present a modified penetrator capabilities model.

4.1 Tags

A tag can be formally defined as:

$$Tag ::= \mathsf{JOIN}\ SID\ CNO$$

A typical tag in a run α will be written as (sid_α, cno). The first part is the session-id of α and the second part is the component number for one of the encrypted facts of α. Tags have to be essentially under every encrypted fact. If an encrypted fact has parts that are encrypted, then a different tag must exist for each of the encrypted facts inside it. It is also useful to define a projection function to derive the session-id and component number from a tag as

$$(t)_s \mathrel{\widehat{=}} sid,\ (t)_c \mathrel{\widehat{=}} cno.$$

Note: As in [102], JOIN and ENCR represent concatenating two data items and encrypting a data item respectively. When two data items $a,\ b$ are to be concatenated, we will write $a\ .\ b$ or (a, b). When a data item a is to be encrypted with a key k, we will write, $\{a\}_k$.

4.2 Facts

We use the terms component, message and fact interchangeably in this thesis.[1] Facts are defined as:

$$\begin{aligned}
Fact &::= UF \mid EF \mid \mathsf{JOIN}\ Fact\ Fact\\
UF &::= \mathsf{JOIN}\ Atom\ UF\\
EF &::= \mathsf{ENCR}\ TF\\
TF &::= \mathsf{JOIN}\ Tag\ Fact
\end{aligned}$$

where $UF,\ EF,\ TF$ represent unencrypted, encrypted and tagged fact respectively. This grammar can be further simplified but is more useful in this form for further parts of the proof.

Let uf, ef denote the set of unencrypted and encrypted facts respectively. Let Atom be the set of atomic values (eg. $Alice$, Bob, N_A, $\mathsf{PubKey}(A)$ etc.) assumed to be contained in a protocol. By assuming these atomic values, many different messages can be created by pumping those values into the strings of the language generated by the above grammar.

Note that the grammar requires that the strings in the language generated by using the grammar should obey some rules:

1. Every encrypted component is tagged.
2. If an encrypted component has sub-encrypted components, each of them must have a tag.

Similar to tags, a projection function on tagged facts is also useful. In this case, the projection function can be defined to derive the tag and fact components from a tagged fact. This projection is defined as:

If (t, f) is a tagged fact, then
$$(t, f)_1 \mathrel{\widehat{=}} t,\ (t, f)_2 \mathrel{\widehat{=}} f.$$

4.3 Subfacts

Similar to subfacts defined in section 2.3, subfacts for the facts defined above can also be defined.

Definition 12. *The subfact relation is the smallest relation on facts such that:*

[1]although message is used to mean a collection of facts (or components) sent in a single protocol step.

1. $f \sqsubset f$*;*

2. $f \sqsubset \{tf'\}_{k'}$ *if* $f \sqsubset (tf')_2$*;*

3. $f \sqsubset (f_1, f_2)$ *if* $f \sqsubset f_1 \vee f \sqsubset f_2$*.*

sub-untagged-facts of a tagged fact can also be defined:

$f \sqsubset tf$ if $(t, f) \sqsubset tf$ for some tag t.

4.4 Adapting strand space model with tagged facts

As mentioned earlier, we will use the strand space model of [102] for the proof because it provides a particularly suitable notation for the kind of reasoning required for the proof. However, the results of the proof are quite general and can be applied to various analysis approaches to security protocols such as those in [187, 97, 162] etc. A brief overview of the strand space model to adapt it to deal with facts defined in section 3.1 is given below. There is some new terminology used to suit the present model, while some of the terminology of the original strand space model is maintained.

Definition 13. *A strand is a sequence of communications by either an honest agent or a penetrator involved in a protocol run. It is represented as a sequence of the form* $\langle \pm f1, \pm f2, \ldots, \pm fn \rangle$*,* $f1, \ldots, fn$ *are facts (similar to terms in the original strand space model). A '+' indicates transmission of a fact and '-' indicates reception of a fact. Every node belongs to a unique strand. The set of a nodes is denoted as* $\mathcal{N}$*.*

1. Let n_i, n_{i+1} *be consecutive nodes on the same strand. Then, there exists an edge* $n_i \Rightarrow n_{i+1}$ *in the strand.*

2. If $n_i = +f$ *and* $n_j = -f$ *are nodes belonging to different strands, then there exists an edge* $n_i \rightarrow n_j$*.*

3. $\mathcal{N}$ *together with both the sets of edges* $n_i \Rightarrow n_{i+1}$ *and* $n_i \rightarrow n_j$ *is a directed graph,* $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow) \rangle$*.*

A bundle represents either part or whole of a protocol run. It is an acyclic, finite subgraph of $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow) \rangle$*. Formally, if* $\rightarrow_C \subset \rightarrow$*,* $\Rightarrow_C \subset \Rightarrow$ *and* $(\rightarrow_C \cup \Rightarrow_C)$ *is a finite set of edges, then* $C = (\rightarrow_C \cup \Rightarrow_C)$ *is a bundle if:*

1. If $n_2 \in \mathcal{N}_C$ *has a negative sign for it's fact, there exists a unique* n_1 *such that* $n_1 \rightarrow_C n_2$*;*

2. Whenever $n_2 \in \mathcal{N}_C$ *and* $n_1 \Rightarrow n_2$*, then* $n_1 \Rightarrow_C n_2$*;*

3. C is acyclic.

4.4.1 Unique origination in bundles

Security properties of a protocol are essentially dependent on maintaining the secret values and the values that originate uniquely. We capture this aspect in our adapted strand space model through the following definition.

Definition 14. *Entry point, origination and unique origination in the adapted strand space model are defined as:*

1. *A node n is an entry point to a set of facts S if the term of n is $+f$ for some $f \in S$, and for each node n' preceding n on the same strand the term of $n' \notin S$.*
2. *A fact originates on a node n if n is an entry point to the set $S' = \{f' \mid f \sqsubset f'\}$.*
3. *Similarly, a tagged fact tf will be said to originate on a node n if n is an entry point to the set $S'' = \{tf' \mid tf \sqsubset tf'\}$.*
4. *A fact or a tagged fact is said to be uniquely originating in a bundle if it originates on a unique node of the bundle.*

4.4.2 Honest strands

In this section, we will show how we model honest strands. They are also called regular strands. Similar to [130], we use the concept of a *strand template* to define each role in a protocol. Upon instantiation of these templates, we obtain honest strands.

Consider a set Var representing variables. Elements of this set form the language generated by the following grammar:

$$\begin{aligned}
Var &::= UV \mid EV \mid \mathsf{JOIN}\ Var\ Var \mid \mathsf{APPLY}\ Function\ Var \\
UV &::= AtomVar \mid \mathsf{JOIN}\ AtomVar\ UV \\
EV &::= \mathsf{ENCR}\ TaggedVariable \\
TaggedVariable &::= \mathsf{JOIN}\ TagVar\ Var
\end{aligned}$$

where UV - Unencrypted Variable, EV - Encrypted Variable. $TagVar$ - Tag Variable.

The variables defined by the above grammar may include elements of the set $\mathsf{AtomVar}$. Instantiation of elements of $\mathsf{AtomVar}$ correspond to elements of Atom. The variables termed as Var also consist of a set of *function variables* (similar to those defined in [130]). Function variables are obtained by the application of a *function identifier* on variables, denoted as $\mathsf{APPLY}\ F_n\ Var$, where $\mathsf{F_n}$ denotes the set of function identifiers. The variable APPLY g $(v_1, \ldots, v_n)$ represents the function g applied to the variable, $v = (v_1, \ldots, v_n)$. Obviously, the functions thus obtained would be partial and undefined when applied to arguments of the incorrect type.

Before defining strand templates to suit the facts defined in the previous sections, consider the set Tags and TagVar. We shall define an instantiation function, $\mathsf{ins_t}$, on the set of tag variables, TagVar:

Let,

$$\mathsf{ins}_t : \mathsf{TagVar} \rightarrow \mathsf{Tags}$$

so that, if $tv \in \mathsf{TagVar}$, $\mathsf{ins_t}(tv) = t$ for some $t \in \mathsf{Tags}$.

Also let

$$\mathsf{ins}_v : AtomVar \rightarrow Atom$$

so that if $v \in \mathsf{AtomVar}$, $\mathsf{ins_v}(v) = f$ for some $f \in \mathsf{Atom}$.

Using ins_t and ins_v functions, we can now define an instantiating function, ins :

$$\mathsf{ins} : \mathsf{Var} \rightarrow \mathsf{Fact}$$

so that if $v \in \mathsf{Var}$, $\mathsf{ins}(v) = f$ for some $f \in \mathsf{Facts}$.

The ins function can now be defined to apply on all possible variables as follows:

$$\mathsf{ins}(v) = \begin{cases} \{(\mathsf{ins_t}(tv'), \mathsf{ins_v}(v'))\}_{\mathsf{ins_v}(k')} & \text{if } v \in \mathsf{ev} \text{ such that } v = \{(tv', v')\}_{k'} \\ (\mathsf{ins}(v_1), \mathsf{ins}(v_2)) & \text{if } v_1, v_2 \in \mathsf{Var} \text{ and } v = (v_1, v_2). \\ g(\mathsf{ins}(v')) & \text{if } v \text{ is a function variable such that} \\ & v = g(v'), g \in \mathsf{F_n} \text{ and } v' \in \mathsf{Var}. \end{cases}$$

The definition of a strand template in this model can now be formalized:

Definition 15. *Let* $\{v_1, \ldots, v_n\} \in \mathsf{Var}$ *be a set of variables in a template* St*. Let* $\{f_1, \ldots, f_n\} \in \mathsf{Facts}$ *be a set of facts in a strand* S*.* St *is said to be a strand template for strand* S *iff for each node* $\langle a, v\rangle, v \in \mathsf{Var}(st), a \in \{+, -\}$*, there exists a corresponding node* $\langle b, f\rangle, f \in \mathsf{Facts}(s), b \in \{+, -\}$ *such that,* $\mathsf{ins}(v) = f$ *and* $a = b$*.*

In other words, if $St = \langle\langle a_1, v_1\rangle, \ldots, \langle a_n, v_n\rangle\rangle$ and $S = \langle\langle b_1, f_1\rangle, \ldots, \langle b_n, f_n\rangle\rangle$, then, $\forall v_i \in \{v_1, \ldots, v_n\} \wedge a_i, b_i \in \{+, -\} \wedge f_i \in \{f_1, \ldots, f_n\} \wedge i \in \{1, \ldots, n\}$, $\mathsf{ins}(v_i) = f_i$ and $a_i = b_i$.

All strands representing an execution of a particular role can be formed by instantiating the free variables of the corresponding template, i.e., by substituting the tag variables with tags and the atomic variables with atoms.

For example, let *temp* be a strand template. Then the role played by b in the Woo and Lam protocol [272] (shown in section 1.4) can be defined by the following sequence of variables:

$$temp = \langle -(a), +(n_b), -(x), +(\{t_2, a, b, x\}_{sh_{bs}}), -(\{t_3, a, b, n_b\}_{sh_{bs}})\rangle$$

This strand template uses five variables: a, b, s, n_b and x. (The first encrypted component in the protocol, which b cannot decrypt, is represented by a variable x, indicating that b is not expected to decrypt this component according to the protocol). Also, it uses two tag variables: t_2, t_3.

Let Alice and Bob be represented by A and B respectively. A typical execution by Bob in a run with Alice, using a session-id of, say, '1120' would look like (assuming that Bob is honest):

$$\begin{aligned}\mathsf{ins}(temp) = \langle\, & -(A),\\ & +(N_B),\\ & -(X),\\ & +(\{(1120,2),A,B,X\}_{sh_{BS}}),\\ & -(\{(1120,3),A,B,N_B\}_{sh_{BS}}).\rangle\end{aligned}$$

This is obtained by using the instantiation function ins, where ins is defined as:

$$\mathsf{ins}(a) = A,\ \mathsf{ins}(b) = B,\ \mathsf{ins}(s) = S, \mathsf{ins}(n_b) = N_B,$$
$$\mathsf{ins}(x) = \{(1120,1),A,B,N_B\}_{sh_{BS}} (= X).$$

It is important here to note that the function ins can be defined to map to a new set of values each time. This captures the aspect of different protocol runs containing different values. It is also interesting to see for which mapping of the ins function we will be able to obtain an attack on a protocol. In fact, some have aimed at showing that it is impossible to find an instantiation function that will model an attack in their scheme [130].

The definition of a strand template also includes the implicit assumption that when an honest agent first sees an atomic variable (untagged) in a message that it receives, it will accept any value for the variable.

4.4.3 Correct-tagging

An encrypted fact is said to be well-tagged if the tag component of the fact has the correct session-id and component number in it. Therefore, in a sense, the encrypted fact is being generated and sent in the expected context. We capture this using the formalizations that we present below.

Definition 16. *Let $\Pi = \{\pi_1, \ldots, \pi_m\}$ be a set of strand templates representing a protocol. Let $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ be a set of strands such that,* $\mathsf{ins}(\pi_i) = \sigma_i$ *for $i = 1, \ldots, m$. Let α be a subset of Σ representing a protocol run. Then a unique component number can be assigned to each encrypted fact in the set of encrypted facts in α using a bijective function* CNo *defined as,*

$$\mathsf{CNo} : \mathsf{ef} \times \Sigma^* \rightarrow \mathsf{cno}$$

such that,

$\forall \mathsf{ef} \in \alpha \wedge \forall f_1, f_2 \in \mathsf{ef}, \exists c_1, c_2 \in \mathsf{cno} \,\centerdot\, CNo(f_1, \alpha) = \mathsf{CNo}(f_2, \alpha)$ *iff* $f_1 = f_2$

We can also impose an ordering for the component numbers assigned to elements of ef using a precedence relation, $\prec$.

Definition 17. *Let S be a set of encrypted facts. Then $\prec_S$ is a partial order. i.e. a reflexive, anti-symmetric and transitive closure of S. Every non-empty subset of S has $\prec_S$-minimal and $\prec_S$-maximal members.*

The relation $\prec_S$ is a subset of $S \times S$. (when the set S is understood, we will write $\prec$). Now let ef be the set of encrypted facts in α. For each $f \in \mathsf{ef}$, a component numbering order can be enforced such that

$\forall f_1, f_2 \in \mathsf{ef}$, if $f_1 \prec f_2$ then $\mathsf{CNo}(\alpha, f_1) < \mathsf{CNo}(\alpha, f_2)$.

Also, $\mathsf{CNo}(\alpha, \prec -minimal(\mathsf{ef})) = 1$ and $\mathsf{CNo}(\alpha, \prec -maximal(\mathsf{ef})) = n$, if the cardinality of $\mathsf{ef} = n$.

Also let,

$$\mathsf{SId} : \Sigma^* \rightarrow \mathsf{sid}$$

such that, for $\alpha \subseteq \Sigma^*$, $\mathsf{SId}(\alpha) \in \mathsf{sid}$ is the session-id for the protocol run, α.

An ideal tag environment, ι can be formally defined on the set of strands Σ representing a protocol run α, such that for any fact $f \in \mathsf{ef}$ in α, $\iota(\alpha, f)$ gives the ideal tag to be used inside f, i.e.

$$\iota : \Sigma^* \times \mathsf{ef} \rightarrow \mathsf{Tag}$$

such that

$$\forall \alpha \subseteq \Sigma^* \wedge \forall f \in \mathsf{ef}, \iota(\alpha, f) = (\mathsf{SId}(\alpha), \mathsf{CNo}(\alpha, f))$$

When the context (protocol run) is understood, we will simply write, $\iota(f)$. The concept of an encrypted fact being correctly tagged can be defined using the above formalization.

Definition 18. *Let $f = \{tf'\}_{k'}$ be a fact in a protocol run α; then well-tagged can be inductively defined on all possible facts as follows*[2]*:*

- *if* $(tf')_2 \in \mathsf{uf}$ *then well-tagged*$(f) \Leftrightarrow (tf')_1 = \iota(f)$*;*

[2]Recall that subscripts “1” and “2” return the tag and fact component respectively from a taggedfact (Sec 3.1)

- *if* $(tf')_2 \in \mathsf{ef}$ *then well-tagged*$(\{tf'\}_{k'})$ *if well-tagged*$((tf')_2) \wedge (tf')_1 = \iota(f)$*;*
- *well-tagged*$(f1, f2) \Leftrightarrow$ $($*well-tagged*$(f1) \wedge$ *well-tagged*$(f2))$ *if* $f1, f2 \in \mathsf{ef}$.

Note that well-tagged is a partial function. Therefore, it is undefined for facts that are not encrypted.

Assumption 3. *There exists an ideal tag environment,* ι *for each set of strands representing a protocol run that is obtained by instantiating a set of strand templates such that all the facts in the protocol run are well-tagged with respect to* ι*.*

In other words, we assume that it is possible to generate a session-id for each protocol run and give each of the encrypted facts in the protocol run a unique component number. Facts of any different protocol run using the same protocol can be compared with ι to see if they are well-tagged.

Assumption 4. *If the fact* f *originates on a regular strand, then well-tagged(*f*).*

This assumption has a couple of facets:

- If an honest agent instantiates an encrypted variable, it instantiates the session-id component of the tag inside it with the session-id of the protocol run in which it intends to send the message.
- Also, it includes the correct component number for the component together with the session-id.

4.4.4 Penetrator strands

Penetrator strands in the original strand space model should also be modified in order to adapt with this model. This is necessary because penetrator strands also form a part of the total strand space in a typical execution of a protocol. The strands of the penetrator in this model is represented by the series of strands below using some possible atomic actions.

The change in the structure of the penetrator strands as compared to the original strand space model is the inclusion of a *Replaying* (**R**) strand. Also, in addition to the assumption that the penetrator has a set of keys $\mathsf{K_P}$ and a set of text messages $\mathsf{T_P}$ available to him, we also assume that he has a set of encrypted facts, $\mathsf{E_P}$, with him that he would have somehow obtained (e.g. by eavesdropping over a network, obtained from a previous run in which he was a legitimate user).

Definition 19. *A penetrator strand is one of the following:*

M	***Text message*** *$\langle +f \rangle$ with $f \in \mathsf{T_P}$.*
F	***flushing*** *$\langle -f \rangle$.*
T	***Tee*** *$\langle -f, +f, +f \rangle$.*
C	***Concatenation*** *$\langle -f_1, -f_2, +f_1f_2 \rangle$.*
S	***Separation*** *$\langle -f_1f_2, +f_1, +f_2 \rangle$.*
K	***Key*** *$\langle +k \rangle$ with $k \in \mathsf{K_P}$.*
E	***Encryption*** *$\langle -k, -f, +\{(t, f)\}_k \rangle$, with $k \in \mathsf{K_P}$.*
D	***Decryption*** *$\langle -k^{-1}, -\{(t, f)\}_k, +f \rangle, k \in \mathsf{K_P}$.*
R	***Replaying*** *$\langle +f \rangle, f \in \mathsf{E_P}$.*

The replaying strand captures the action of a penetrator sending an encrypted component that he acquired from another protocol run or within the same protocol run but in a different message (replay).

Note that we consider not only replaying of encrypted components, but also replaying of unencrypted components. In fact, we allow the penetrator to replay a message of any type into a message of a different (expected) type, e.g., an *Atom* in place of another *Atom*, an *Atom* in place of another encrypted component, an encrypted component in place of an encrypted component and so on. For example, sending a message, $f_1.f_2$ with $f_1 \in \mathsf{uf}$ and $f_2 \in \mathsf{ef}$ in place of $f_3 \in \mathsf{uf}$ can be constructed as a sequence of **M** and **R** strands.

However, this does not restrict the generality of the scheme. As we will show in the proof, the scheme also prevents all such type flaw attacks too, due to component numbering. The session-id helps in preventing all type-flaw attacks (which, as we discussed in section 2, are one class of replay attacks) that occur not only in the same run or a different run of the same protocol, but also those that occur from a different run using a different protocol.

Lemma 5. *Every ill-tagged fact originates on an **E** or an **R** strand.*

Proof. According to assumption 1, ill-tagged facts do not originate on honest strands. The only possible strands for the origin of a fact are, **M, K, E** and **R** strands. In the case of **M, K** strands there is no tagging. In the case of **E** strand, it may or may not be well-tagged (we do not restrict the penetrator to put correct tags inside encryptions, but arbitrary tags). That leaves us with the **R** strands. Since these involve replaying of old message components, they would be ill-tagged.

4.5 Security properties and attacks

In this section, we consider the security properties of protocols. We have already given generic definitions of these properties in section 2.3.3. These are applicable to strands instantiated using strand templates that do not involve tagging, using an instantiation function, sub. Here we would redefine the security properties within the adapted strand space model, for strands that are instantiated, by using the instantiation function ins on strand templates that have tagged variables.

4.5.1 Secrecy

Similar to the earlier definition of secrecy, this definition says that when certain keys have not been compromised in a strand of minimal length, there is still a breach of security if an honest strand has entry points to a set of secrets. If there is no such honest agent having entry points to a set having secret elements, then a penetrator also cannot "say" an element in the set since a penetrator can only send an element that he can deduce after hearing that element once.

Definition 20. *Let temp be the template for some role; let $v \in \mathsf{uv}$ be a variable of temp, intended to remain secret; let h be a positive integer; and let Keys be a set of function variables. A failure of secrecy exists when each of the following holds:*

1. *There is a strand $s = \mathsf{ins}(temp)$ with C-height at least h (i.e. at least the first h messages of s appear in the bundle C).*
2. *$\forall k \in \mathsf{Keys}$. $\mathsf{ins}(k) \notin \mathsf{K_P}$. (No private key is visible to the penetrator)*
3. *There is a node in C with label $+\mathsf{ins}(v)$ with $v \in \mathsf{uv}$.*

The parameters of this definition are: $temp, v, \mathsf{Keys}$ and h; for a given protocol one would be interested to see if this property holds for some particular values of this parameters. (i.e. finding an instantiation function that would instantiate these variables to satisfy each of the three conditions above).

4.5.2 Authentication

We now redefine the property of authentication. Similar to the earlier definition in section 2.3.3, we would expect that whenever there is a strand s_1 of r_1, there should be a "corresponding" strand s_2 of r_2; these strands should agree upon the identities of the agents involved, and possibly upon the values of some other variables. We would capture this aspect by specifying that the strands should agree on the values of all variables from set X.

Definition 21. *Let $temp1$ and $temp2$ be templates for two roles; let X be a set of variables of those templates; let $h1$ and $h2$ be positive integers; and let Keys be a set of function templates. A failure of authentication exists when each of the following holds:*

1. *There is a strand $s1 = \mathsf{ins1}(temp1)$ with C-height at least $h1$.*
2. *$\forall k \in Keys$. $\mathsf{ins1}(k) \notin \mathsf{K_P}$.*
3. *There is no strand $s2 = \mathsf{ins2}(temp2)$ with C-height at least $h2$ such that $\forall x \in \mathsf{X}$. $\mathsf{ins1}(x) = \mathsf{ins2}(x)$.*

This definition is parameterized by $temp1, temp2, \mathsf{X}, h1, h2$ and Keys. (See section 3.4.2 for the definition of the instantiation function ins).

Chapter 5

Transforming Bundles

5.1 Overview

We focus on transforming bundles in this chapter. We shall show that, given a bundle C, we can rename all the tags in C so that the resulting bundle has facts, all of which are well-tagged. In particular, if every protocol execution is a bundle then the transformation corresponds to a scenario where the resulting bundle still reflects the same protocol but under a different execution with possibly different penetrator strands. The honest strands however are preserved. We shall show this by proving that the regular strands in the original bundle get transformed into regular strands of the transformed bundle (instantiation of some strand templates).

We start off by defining such a transformation function (ψ) and show that such a ψ can always be constructed.

5.2 The transformation function, ψ

The definition below states the required properties of ψ:

Definition 22. *Given a bundle C, executed in an ideal tag environment ι, we define*

$$\psi : Fact \rightarrow Fact$$

to be a transformation function that transforms C as a well-tagged bundle if:

1. *ψ preserves unencrypted facts: if $\psi(f) = f$ if $f \in \mathsf{uf}$.*
2. *ψ returns well-tagged terms: well-tagged($\psi(f)$).*
3. *ψ is the identify function over well-tagged terms: if well-tagged(f) then $\psi(f) = f$.*
4. *ψ distributes through encryptions*

$$\psi(\{(t, f)\}_k) = \{(\iota(f), \psi(f))\}_k$$

5. *ψ distributes through concatenations*

$$\psi(f1, f2) = (\psi(f1), \psi(f2))$$

6. *When ψ is applied to an ill-tagged fact f of C, it produces a fact that has a new tag, i.e. a fact that has a tag not in common with $\psi(f')$ for any other fact f' of C*

$$\forall f \in \mathsf{ef}, \neg \textit{well-tagged}(f) \wedge f' \sqsubset \psi(f) \Rightarrow \forall f'' \in \mathsf{ef}, (f')_1 \neq \psi(f'')_1$$

This establishes an injectivity property for ψ over facts of C.

The following lemma proves that such a ψ can always be found.

Lemma 6. *For any given bundle C in an ideal tag environment ι, it is possible to define some transformation function ψ for C.*

Proof. The method we give below gives a recipe for constructing a transformation function ψ defined in definition 22. Let there be a fact f. We define how the transformation would be done on all possible smallest sub-facts of f before defining it on f itself. We will consider the possible sub facts of f, case by case, until we define ψ over all possible subfacts.

1. If $f \in \mathsf{uf}$, lift the transformation function on f for condition 1: $\psi(f) = f$.
2. If $f = \{(t', f')\}_{k'} \in \mathsf{ef}$ and well-tagged(f) then define $\psi\{(t', f')\}_{k'} = \{(t', f')\}_{k'}$ for condition 3.
3. If $f = \{(t', f')\}_{k'} \in \mathsf{ef}$ and $\neg$well-tagged(f) then define $\psi\{(t', f')\}_{k'} = \{(t'', f')\}_{k'}$ so that $t'' = \iota(f')$ for condition 3.
4. If $f = (f1, f2)$, then define $\psi(f) = (\psi(f1), \psi(f2))$ (for condition 4).

The only assumption is that we can always change an ill-tagged fact to a well-tagged fact. This is not very difficult. For example, honest agents always receive two kinds of encrypted messages.

1. Those they can decrypt; and,
2. Those that they are not expected to decrypt (according to the protocol). Let these be named as some set $\mathsf{X_H}$.

Our transformation function does not change the tags inside those messages that the honest agent can decrypt (The reasoning is trivial; an honest agent does not accept an encryption that has an invalid tag. This was part of our initial assumptions). Informally speaking, if an honest agent is willing to accept facts that have a subfact from set $\mathsf{X_H}$ that is ill-tagged, then they should accept any value in it's place. Hence, we change the tags inside all ill-tagged facts so that the facts are now well-tagged. This corresponds to our ability to change the *sid* to that of the *sid* of the protocol run and the *cno* to that of the *cno* of that encrypted component by doing a look up on the set cno of that protocol run. Note also that ψ is a partial function. For example, when the penetrator sends an encrypted fact that has a different expected type, the format of the message may not correspond to any element in ef. Hence, ψ is undefined for all those encrypted facts that are not contained in ef. □

5.3 Regular strands

In this section we show that if S is a regular strand then $\psi(S)$ is also a regular strand. By definition, if S is a regular strand, then it must be an instantiation of a strand template *temp* under an instantiation function ins such that $S = \mathsf{ins}(temp)$. Now consider another strand S' formed by instantiating *temp* using another instantiation function ins' defined as

$$\mathsf{ins}'(v) = \psi(\mathsf{ins}(v))$$

so that, $\mathsf{ins}'(v) = \mathsf{ins}(v),\ \forall v \in \mathsf{uv}$.

Since S' is merely another strand instantiated by using a different instantiation funciton on the same strand template, by definition it would be a regular strand too. We will show in the following lemma that S' can be obtained by transforming each of the facts in S using ψ.

Lemma 7. *Let temp*, $\psi, \mathsf{ins}, \mathsf{ins}'$ *be as above; Then,*

$$\psi(\mathsf{ins}(temp)) = \mathsf{ins}'(temp).$$

Proof. Let v be a variable in *temp*. We show that $\psi(\mathsf{ins}(temp)) = \mathsf{ins}'(temp)$ by doing a case analysis of all the possible forms that v can take in *temp*.

- Case v is an unencrypted variable, i.e. $v \in \mathsf{uv}$; then:

$$\begin{aligned}
\psi(\mathsf{ins}(v)) &= \psi(\mathsf{ins}'(v)) && \text{(from above, } \mathsf{ins}'(v) = \mathsf{ins}(v), \forall v \in \mathsf{uv}) \\
&= \mathsf{ins}'(v) && \text{(from condition 1 of definition 22.)}
\end{aligned}$$

- Case v is a function application, e.g. $v = g(v_1, \ldots, v_n),\ v_1, \ldots, v_n \in \mathsf{AtomVar}$; then:

$$\begin{aligned}
\psi(\mathsf{ins}(v)) &= \psi(\mathsf{ins}(g(v_1, \ldots, v_n))) \\
&= \psi(g(\mathsf{ins}(v_1, \ldots, v_n))) && \text{(from definition of } \mathsf{ins}) \\
&= \psi(g(\mathsf{ins}(v_1), \ldots, \mathsf{ins}(v_n))) && \text{(again from definition of } \mathsf{ins}) \\
&= \psi(g(\mathsf{ins}'(v_1), \ldots, \mathsf{ins}'(v_n))) && \text{(from above)} \\
&= \psi(g(\mathsf{ins}'(v_1, \ldots, v_n))) && \text{(from definition of } \mathsf{ins}) \\
&= \psi(\mathsf{ins}'(g(v_1, \ldots, v_n))) \\
&= \psi(\mathsf{ins}'(v)) \\
&= \mathsf{ins}'(v) && (\psi(\mathsf{ins}(v)) = \mathsf{ins}'(v),\ \forall v \in \mathsf{uv})
\end{aligned}$$

- Case v is an encrypted fact, e.g. $v = \{(tv', v')\}_{kv'}$; then:

$$\begin{aligned}
\psi(\mathsf{ins}(v)) = \; & \psi(\{(\mathsf{ins_t}(tv'), \mathsf{ins_v}(v'))\}_{\mathsf{ins_v}(kv')}) \\
= \; & \text{(by condition 3 of definition 22 and since well-taged}(\mathsf{ins}(v))\text{ from assumption 2)} \\
& \{(\mathsf{ins_t}(tv'), \mathsf{ins_v}(v'))\}_{\mathsf{ins_v}(kv')} \\
= \; & \text{(again by assumption 2, and since well-tagged}(\mathsf{ins}'(v)), \mathsf{ins'_t}(tv') = \mathsf{ins_t}(tv')) \\
& \{(\mathsf{ins'_t}(tv'), \mathsf{ins_v}(v'))\}_{\mathsf{ins_v}(kv')} \\
= \; & \text{(from definition of } \mathsf{ins} \text{ and } \mathsf{ins'_v}(v') = \mathsf{ins_v}(v'), \mathsf{ins'_v}(kv') = \mathsf{ins_v}(kv')) \\
& \mathsf{ins}'(v)
\end{aligned}$$

- Case v is a pair, e.g. $v = (v_1, v_2)$; then,

$$\begin{aligned}
\psi(\mathsf{ins}(v)) = \; & \psi(\mathsf{ins}(v_1), \mathsf{ins}(v_2)) && \text{(by definition of } \mathsf{ins}) \\
= \; & (\psi(\mathsf{ins}(v_1)), \psi(\mathsf{ins}(v_2))) \\
= \; & (\mathsf{ins}'(v_1), \mathsf{ins}'(v_2)) && \text{(from above results)} \\
= \; & \mathsf{ins}'(v_1, v_2) \\
= \; & \mathsf{ins}'(v)
\end{aligned}$$

□

5.4 Penetrator strands

In this section, we present the continuing saga of transforming a bundle. Any given bundle can contain both regular strands and penetrator strands. In the previous section we have shown that transforming regular strands in a bundle correspond to some other regular strands of the same protocol without altering either the message format or the strand structure. In this section we show similarly how we transform penetrator strands in a given bundle to contain no ill-tagged facts. We will see that we wouldn't need to change the strand structure even in this case. We will just change the tags inside all facts that are ill-tagged so that now they are well-tagged. Intuitively, penetrator strands may use some facts that the honest agent doesn't decrypt (the set X_H). These may or may not be well-tagged. However, for example, if the honest agent is tricked into accepting an *Atom* in place of an encryption, we do not transform the fact. Instead, we lift the ψ since it is an unencrypted fact. This will not alter the result of the proof and will be effective in preventing all type-flaws as will be shown later. As before, we assume that he has some text messages, $\mathsf{T_P}$, keys, $\mathsf{K_P}$, and some encrypted facts, $\mathsf{E_P}$ with him.

M ***Text message*** Let $S = \langle +x \rangle$ with $x \in \mathsf{T_P}$. Define $S' = \langle +\psi(x) \rangle$, which is an **M** strand because $\psi(x) = (x)$ when $x \in \mathsf{T_P}$.

F ***Flushing*** Let $S = \langle -f \rangle$. Define $S' = \langle -\psi(f) \rangle$, which is an **F** strand.

T ***Tee*** Let $S = \langle -f, +f, +f \rangle$. Define $S' = \langle -\psi(f), +\psi(f), +\psi(f) \rangle$, which is a **T** strand.

C ***Concatenation*** Let $S = \langle -f_1, -f_2, +f_1 f_2\rangle$. Define $S' = \langle -\psi(f_1), -\psi(f_2), +\psi(f_1, f_2)\rangle$ which is a valid concatenation strand because

$$\psi(f_1, f_2) = (\psi(f_1), \psi(f_2))$$

by condition 5 of definition 22.

S ***Separation*** Let $S = \langle -f_1 f_2, +f_1, +f_2\rangle$. Define $S' = \langle -\psi(f_1, f_2), +\psi(f_1), +\psi(f_2)\rangle$ which is a valid separation strand, again by condition 5 of definition 22.

K ***Key*** Let $S = \langle +k\rangle$ with $k \in \mathsf{K_P}$. Define $S' = \langle +\psi(k)\rangle = \langle +k\rangle$, which is a **K** strand.

E ***Encryption*** Let $S = \langle -k, -f, +\{(t, f)\}_k\rangle$ with $k \in \mathsf{K_P}$. Define

$$S' = \langle -\psi(k), -\psi(f), +\psi(\{(t, f)\}_k)\rangle$$

which is a valid encryption strand because $\psi(\{(t, f)\}_k) = \{\iota(f), \psi(f)\}_k$ by condition 4 of definition 22.

D ***Decryption*** Let $S = \langle -k^{-1}, -\{(t, f)\}_k, +f\rangle, k \in \mathsf{K_P}$. Define

$$S' = \langle -\psi(k^{-1}), -\psi(\{(t, f)\}_k), +\psi(f)\rangle,$$

which is a valid decryption strand because, $\psi(\{(t, f)\}_k) = \{(\iota(f), \psi(f))\}_k$ by condition 4 of definition 22.

R ***Replaying*** Let $S = \langle +f\rangle, f \in \mathsf{E_P}$. Define $S' = \langle +\psi(f)\rangle$ which is a valid replaying strand because $\psi(f)$ is merely well-tagged without any additional change in the message.

One special case here concerns when the penetrator receives a well-tagged fact and sends another fact in it's place, either by replaying or by "retagging": i.e. a combination of (a) **F** and **R** strands: $\langle -f, +f'\rangle$ with $f' \in \mathbf{E_P}$ or (b) **D** and **E** strands: $\langle -f, +f'\rangle$ $(\langle -\{(t_1, f_1)\}_{k_1}, -k_1, +f_1,$
$+\{(t_2, f_2)\}_{k_2}, -k_2, +f_2\rangle)$, with $k_1, k_2 \in \mathbf{K_P}, f = \{(t_1, f_1)\}_{k_1}$ and $f' = \{(t_2, f_2)\}_{k_2}$.

This can be visualized as in figure 5.1. In this case, we change the strand structure by removing all the intermediate nodes between $+f$ and $-f'$ as in figure 5.2 and replace the entire structure with $\psi(f) \rightarrow \psi(f')$.

5.5 Preserving unique origination in bundles

Thus far we have described how a given bundle C can be transformed into another similar bundle C' in which all the facts are well-tagged. We have shown that it is possible to achieve the transformation without altering the strand structure anywhere. Our transformation merely transforms ill-tagged facts (ones which are produced by the penetrator and not checked by the honest agents) into well-tagged ones. In essence, for every edge $+n \rightarrow -n$ in C, we create a similar edge $+\psi(n) \rightarrow -\psi(n)$ in C'. In this section, we show that this transformation preserves unique origination in bundles.

Note that we have applied the transformation function only on tagged facts and only on the tag component of a tagged fact. Also, we did not change the strand structure or introduce new nodes anywhere.

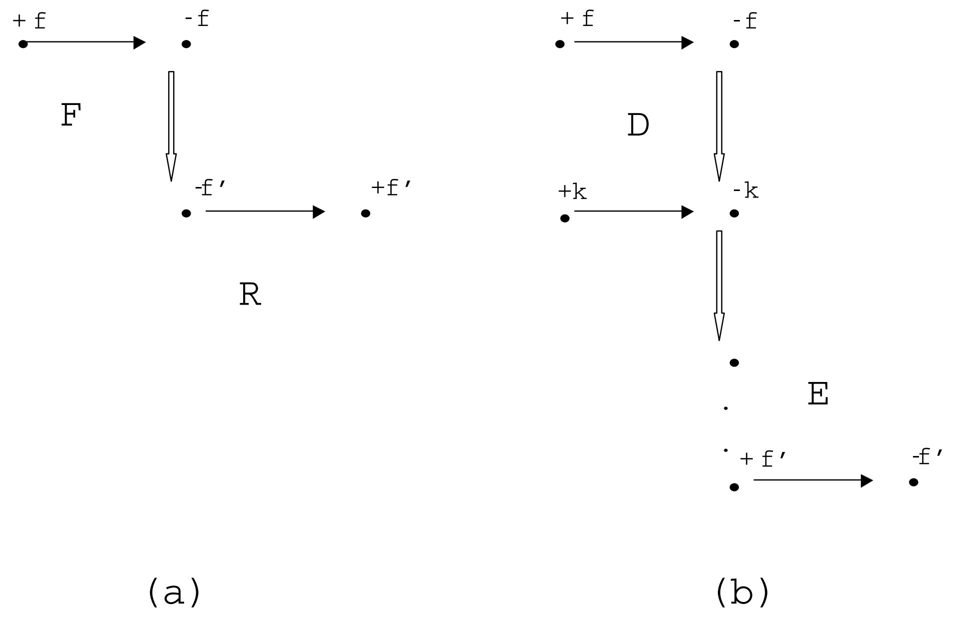

Figure 5.1: Penetrator strands combining (a) **F**, **R** strands (b) **D**, **E** strands

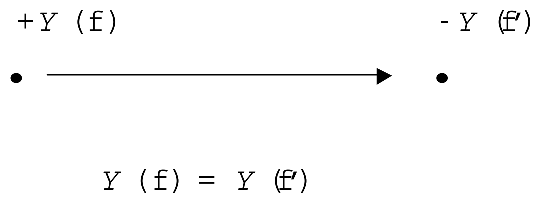

Figure 5.2: Replacing strands in Fig 5.1 using ψ

Lemma 8. *Let C be the original bundle and C' be the transformed bundle such that, $\psi(C) = C'$. If f_0 is a fact, uniquely originating in C, then f_0 also originates uniquely in C'.*

Proof. The only possible places of origin of a fact in the penetrator strand are:

1. **M** strand. In this case, since $\psi(f_0) = f_0$, $\forall f \in \mathsf{uf}$, unique origination is preserved when C is transformed into C';

2. **K** strand. In this case too, the same applies as that of the **M** strand;

3. **R** strand. From the definition of ψ, transforming does not change the fact component of a tagged fact. Also, by the injectivity property of ψ (condition 6 of definition 22), there is no duplication of tags.

Therefore, unique origination is preserved when C is transformed into C' using ψ. □

Chapter 6

Proof

6.1 Overview

In this chapter, we prove our main claim - the tagging scheme prevents all replay attacks. We use the results from previous chapters as the basis for the proof. We follow an indirect approach for our proof. We will show that a protocol is secure under the tagging scheme if it is secure in the absence of replay attacks. Note that we do not intend to prove that there cannot be undetected replays in a protocol run. We only prove that no such replays can aid in attacking the protocol. For this purpose, we chose to identify the attributes of a replay and the correctness properties of a protocol and combine them both to constitute a replay attack.

The focus then is on showing that no attacks exist that use replays if all the honest agents adopt the tagging scheme. To be precise, we show that:

> If a protocol is secure in the absence of replays, then it is secure under the tagging scheme even in the presence of replays.

Put in other words, the tagging scheme prevents all attacks arising out of replays. We turn this around and show that:

> If there is an attack on a protocol under the tagging scheme, there is also an attack on the protocol when adapting the tagging scheme with all tagged messages correctly tagged.

In the previous chapter, we have shown how we can obtain a well-tagged bundle from any given bundle. We have used a transformation on a given bundle and changed all ill-tagged facts to well-tagged facts. We have not changed the strand structure at any place including penetrator strands. We have also shown that the well-tagged bundle preserves unique origination. Those concepts can be summarized in the following theorem.

Theorem 9. *If C is a bundle, then C can be transformed using a transformation function, ψ, into a bundle C' such that:*

- *C' contains the set of facts in C, all of which transform into well-tagged facts using ψ;*
- *C' contains the same honest strands as in C;*
- *C' contains penetrator strands similar to those in C except for a change in tags for some of the tagged facts;*
- *C' has the same strand structure as that of C with facts uniquely originating if they were uniquely originating in C.*

Proof. The results in each of the sections in chapter 5 can be regarded as the proof for each of the statements laid in the theorem. □

In the following sections we prove that if there is an attack on C, there is a corresponding attack on C'. We prove this by considering an attack to mean a failure to achieve a desired goal at the end of the protocol run, e.g. secrecy, authentication (we have defined both in chapter 3), non-repudiation, fairness, anonymity and so on. We shall however, prove it only for the first two properties. Other properties can be similarly considered and proven.

6.2 Secrecy

Theorem 10. *Let C be a bundle and C' be a well-tagged bundle obtained by transforming C. i.e. $\psi(C) = C'$. If there is a failure of secrecy in C, there is also a failure of secrecy in C'.*

Proof. Following definition 20, suppose there is a failure of secrecy in C as follows:

1. There is a strand $s = \mathsf{ins}(temp)$ with C-height at least h; and,
2. $\forall k \in \mathsf{Keys} \cdot \mathsf{ins}(k) \notin \mathsf{K_P}$; and,
3. There is a node n with label $+\mathsf{ins}(v)$.

We show that there is a corresponding attack in C'. Let instantiation function ins' be defined as in section 5.3:

$$\mathsf{ins}'(v) = \psi(\mathsf{ins}(v)).$$

According to def. 20, if there is a failure of authentication in C' then:

1. There is a strand $s' = \mathsf{ins}'(temp) = \psi(\mathsf{ins}(temp))$ with C'-height at least h, corresponding to s from the way we have constructed the honest strands of C'.
2. $\forall k \in \mathsf{Keys} \cdot \mathsf{ins}'(k) \notin \mathsf{K_P}$ because $\mathsf{ins}'(k) = \mathsf{ins}(k)$ for such k.

3. The node corresponding to n will have label $+\mathsf{ins}'(v)$ $(= \psi(\mathsf{ins}(v)))$.

   However, from the definition of ins, we have $\psi(\mathsf{ins}(v)) = \mathsf{ins}(v)\ \forall v \in \mathsf{uv}$. Hence, there is a node $+\mathsf{ins}'(v)$ in C' as well and therefore a failure of secrecy in C'.

□

6.3 Authentication

Theorem 11. *Let C be a bundle and C' be a well-tagged bundle obtained by transforming C, i.e. $\psi(C) = C'$. If there is a failure of authentication in C, there is also a failure of authentication in C'.*

Proof. Following definition 21, suppose there is a failure of authentication in C such that:

1. There is a strand $s1 = \mathsf{ins1}(temp1)$ with C-height at least $h1$; and,
2. $\forall k \in \mathsf{Keys}\ \cdot\ \mathsf{ins1}(k) \notin \mathsf{K_P}$; and,
3. There is no strand $s2 = \mathsf{ins2}(temp2)$ with C-height at least $h2$ such that $\forall x \in X\ .\ \mathsf{ins1}(x) = \mathsf{ins2}(x)$.

We show that there is a corresponding attack in C'. Let instantiation $\mathsf{ins1}'$ be defined as in section 5.3

$$\mathsf{ins1}'(v) = \psi(\mathsf{ins1}(v)).$$

According to def 21, if there is a failure of authentication in C' then

1. There is a strand $s1' = \mathsf{ins1}'(temp1) = \psi(\mathsf{ins1}(temp1))$ with C'-height at least $h1$, corresponding to $s1$, from the way we have constructed the honest strands of C'.
2. $\forall k \in \mathsf{Keys}\ \cdot\ \mathsf{ins1}'(k) \notin \mathsf{K_P}$ because $\mathsf{ins1}'(k) = \mathsf{ins1}(k)$ for such k.
3. There is no strand $s2' = \mathsf{ins2}'(temp2)$ with C'-height at least $h2$ such that $\forall x \in \mathsf{X}\ \cdot\ \mathsf{ins1}'(x) = \mathsf{ins2}'(x)$. Suppose there were such an $s2'$; then, by the way we have constructed the honest strands in C', $s2'$ would correspond to some strand $s2'' = \mathsf{ins2}''(temp2)$ with C'-height at least $h2$ such that

   $$\forall v \in \mathsf{Var}\ \cdot\ \mathsf{ins2}'(v) = \psi(\mathsf{ins2}''(v))$$

   But then we would have for every $x \in \mathsf{X}$

   $$\begin{aligned} \psi(\mathsf{ins1}(x)) = &\ \mathsf{ins1}'(x) \\ = &\ \mathsf{ins2}'(x) \\ = &\ \psi(\mathsf{ins2}''(x)). \end{aligned}$$

However, this contradicts part 3 of the above definition, since we have $\mathsf{ins1}(x) = \mathsf{ins2}''(x)$ due to the injectivity property of ψ (condition 6 of definition 22).

□

6.4 An example

In this section, we shall show that the results obtained through the proof can be applied directly to protocols which are vulnerable to replay attacks. We prove this by showing that if a protocol is executed under the assumption that there is no replay of messages and is secure then it is also secure under the tagging scheme in the presence of replays. We take the example of the Needham-Schroeder Public-Key Protocol that we discussed at various places in this thesis to serve to illustrate this. We have shown how this protocol was attacked using various techniques, such as, man-in-the-middle, type-flaw, multi-protocol and interleaving attacks. Our explanation should now serve to reason about the efficacy of the tagging scheme in preventing all those attacks which involve replaying messages.

The two roles in this protocol can be defined by the strand templates:

$$\begin{aligned} &\mathit{Init} \mathrel{\widehat{=}} \\ &\quad \langle +\{t1, na, a\}_{PK(b)}, \\ &\quad\;\; -\{t2, na, nb, a\}_{PK(a)}, \\ &\quad\;\; +\{t3, nb\}_{PK(b)} \ \rangle, \\ \\ &\mathit{Resp} \mathrel{\widehat{=}} \\ &\quad \langle -\{t1, na, a\}_{PK(b)}, \\ &\quad\;\; +\{t2, na, nb, a\}_{PK(a)}, \\ &\quad\;\; -\{t3, nb\}_{PK(b)} \ \rangle. \end{aligned}$$

Thayer *et al.* in [102] establish a number of properties of this protocol. These established properties assume some conditions. For example, they assume that there exist no type flaws in the messages, no replay of messages from a different protocol and so on. Some are implicit and some are explicit. However, we will now show that the result we were able to achieve guarantees that a protocol remains secure even when all those assumptions are dropped if the tagging scheme is adopted.

As an example, consider Proposition 5.2 in [102] According to this proposition, a 'responder's guarantee' is: Let there be a bundle in which there is a responder's strand $s1 = \mathsf{ins}(\mathit{Resp})$ such that $\mathsf{ins1}(Pv(a)) \notin \mathsf{K_P}$. Then there exists a corresponding initiator's strand $s2 = \mathsf{ins2}(\mathit{Init})$ such that $\mathsf{ins1_v}$ and $\mathsf{ins2_v}$ agree on a, b, na and nb (for $1 \leq i \geq 3$, $\mathsf{ins1_t}(ti) = \mathsf{ins2_t}(ti) = \phi$). In other words, authentication is guaranteed w.r.t the definition we have given as definition 21: There is no failure of authentication if $temp1 = \mathit{Resp}, temp2 = \mathit{Init}, X = \{a, b, na, nb\}, h1 = 3, h2 = 3$ and $Keys = \{Pv(a)\}$.

We can use our main result to show that this protocol achieves this property even in presence of any of the replay attacks we have given in chapter 2. In particular, if *Alice* and *Bob* establish a session with session-id 'sid_α', then the strand templates can be instantiated using $\mathsf{ins1}$ and $\mathsf{ins2}$ defined as

$$\mathsf{ins1_t}(t1) = (sid_\alpha, 1), \mathsf{ins1_t}(t2) = (sid_\alpha, 2), \mathsf{ins1_t}(t3) = (sid_\alpha, 3)$$

and

$$\begin{aligned}\mathsf{ins1_v}(na) = \mathsf{ins2_v}(na) &= N_{Alice},\\ \mathsf{ins1_v}(nb) = \mathsf{ins2_v}(nb) &= N_{Bob},\\ \mathsf{ins1_v}(PK(a)) = \mathsf{ins2_v}(PK(a)) &= PK(Alice),\\ \mathsf{ins1_v}(PK(b)) = \mathsf{ins2_v}(PK(b)) &= PK(Bob).\end{aligned}$$

An authentication guarantee of the protocol was already established by proposition 5.2 in [102] (with $t1 = t2 = t3 =$ null). We can immediately apply our main result to prove that this guarantee is maintained even in presence of replays. Particularly, the protocol remains secure in the presence of replays when

$$t1 = (sid_\alpha, 1), t2 = (sid_\alpha, 2), t3 = (sid_\alpha, 3)$$

from theorems 2 and 3.

Chapter 7

A Communication-Computation Efficient Group Key Algorithm for Large and Dynamic Groups

7.1 Introduction

With the advent of new arenas such as wireless ad-hoc and low powered distributed computing and communication devices, designers of group key encryption algorithms can no longer ignore communication in favor of computation or vice versa. In some environments the power cost of communication may be sufficiently high to warrant low cost communication protocols, whereas in other environments computation cost may be the dominant feature. Consequently, this chapter introduces the Communication-Computation Efficient Group Key protocol (CCEGK), which is a extension of EGK [28] and TGDH [142, 145].

The Communication-Computation Efficient Group Key Algorithm (CCEGK) is a group key management algorithm based upon two preceding group key management algorithms, EGK [28] and TGDH [142, 145]. By extending this previous work, CCEGK considerably improves both communication and computation costs of their related operations. Furthermore CCEGK fully implements, as detailed in this chapter, several methods that are not described by other authors in the literature. For instance, CCEGK fully implements an initialization operation while the TGDH and STR [143, 144] algorithms do not. Next CCEGK presents two mass leave operations, Mass Leave-Balanced, and Mass Leave-Imbalanced (detailed later), while the TGDH algorithm only details a Mass Leave-Imbalanced. The Add operation in CCEGK is distinct from TGDH and similar to EGK; each time the controller sends a message, it sends a message with only one key size, as opposed to TGDH which sends all blinded keys in the group. Our merge method differs from TGDH in a similar manner. Next TGDH and STR's partition (split) operation could be better understood as a mass leave operation; since it is not detailed in literature it will not be compared further. Lastly TGDH and STR do not implement a balance operation, and CCEGK does detail a balance operation later in this chapter.

There are similarities as well. CCEGK's leave and key refresh operations are the same operation as in TGDH, while join and initilize are almost identical to those of EGK and other operations are similar. While CCEGK adopts ideas from both TGDH and EGK, the focus on computation and communication efficiency warrants a distinct algorithm. Table 7.1 details the differences and similarities between CCEGK and TGDH, and CCEGK and EGK.

The goal of this chapter is to present the full details of every operation of CCEGK in a standardized manner. This chapter provides the following material

- We fully describe the main key management operations for CCEGK, including initialization, add, mass add, merge, leave, mass leave, partition, and key refresh. This set of operations provides a basis for full comparison of competing group key protocols.
- Some group key protocols, such as our proposed CCEGK protocol, do not rebalance the binary key tree upon every operation. Therefore, we introduce two separate rebalance schemes, which are not expensive in either computation or communication costs. The rebalance operation is not documented in the literature for EGK, TGDH or STR even though it can greatly affect the overall performance of the protocols over time as the trees become more unbalanced.
- We provide a theoretical analysis of the costs of both computation and communication for each operation for each of the compared protocols. This is beneficial since a complete treatment of all of these operations is not available in the literature. A separate recent study by Challal and Seba reviews a large number of group key protocols, but only compare the initialization costs [74]. The papers introducing STR [143, 144] and TGDH [142, 145], and companion performance paper [38] do not address initialization and partition operations at all (their partition operation is effectively a mass leave operation). We provide this missing information and also illustrate a point of contention with their published results.

7.2 Background

In 1976, Diffie and Hellman introduced a two party key exchange protocol, DH, that allows two participants to create a private key [94] through the use of publicly exchanged messages. This protocol is now at the heart of many two-party secure communication protocols, including SSL [110] and SSH [275]. When we log in to a secure web site, our machine exchanges information with the web server to create a secure session key. That key is used to encrypt subsequent communication, protecting confidential information such as credit card numbers. With the increasing use of network technologies, there are several applications that would greatly benefit from an extended group key exchange algorithm that provides the same protection as DH, but for groups of participants. These applications include conference calls, distributed computation, white-boards and distributed databases, among many others. To ensure secure and reliable communication in these applications, there have been several attempts to create efficient group key protocols for large and dynamic groups based on the DH algorithm [70, 71, 137, 249, 61].

	Key refresh	Initialization	Join	Mass Join	Merge	Leave	Mass Leave
EGK	NA	similar	similar	different	different	different	different
TGDH	same	NA	different	different	different	same	different

Table 7.1: Operation Comparison with CCEGK

These approaches can be divided into two categories of key management, centralized and contributory [28, 248, 37, 38]. In centralized key management, a single member creates keys and securely transmits them to all other group members. The drawback is that a centralized approach requires a trusted sponsor who generates and distributes the keys, leaving a single point of failure. The trusted sponsor cannot be continuously available to support operations in the event of an arbitrary network partition, so this approach is not appropriate for reliable group communication. Some authors have suggested using a distributed approach where the centralized management is distributed among many hosts, increasing the system's fault tolerance and reliability, but still requiring a set of trusted hosts.

The contributory group key management approach is based on the idea that each group member will directly contribute to key management and key generation (each member contributes to the entropy of the key). This alleviates the problem of a single point of failure and trust in a centralized group key management system. In the late of 1990s there were several group key protocols concurrently introduced, such as EGK [28], TGDH [142, 145], and STR [143, 144], based on contributory group key management to improve the costs of communication and computation. EGK, TGDH and STR utilize a binary-tree structure to provide an efficient number of message exchanges, but they provided different algorithms for the key and group management protocols. EGK and TGDH focus more on the cost of computation than on that of communication. STR focuses on improving the cost of communication more than the cost of computation, in keeping with Becker and Willie's original work on efficient communication in group key protocols [61].

7.3 Group Key Exchange and Management Background

The purpose of this paper is truly two-fold: first to introduce a new group-key agreement protocol based on an underlying two-party key exchange and second, to compare our protocol with other protocols that have appeared in the literature. To that aim, we constructed a simulator to evaluate the efficiency of the algorithms, as described by their authors. Additionally, we have analytically compared their algorithmic performance. Before we introduce our protocol, we first provide a summary of the basic concepts and terminology of group key protocols used throughout this paper.

7.3.1 Public Two-Party Key Exchange

A two-party key exchange protocol is a protocol that permits two entities with no prior shared secrets to publicly exchange information so that at the end of the protocol they both possess the same shared secret, and any eavesdropper listening in on the communication is unable to obtain that same secret. After the protocol runs, the participants can derive a session key from the shared secret. Throughout this paper we will use the term group key or secret key to refer to this shared secret and its derivative keys. Diffie and Hellman [94] introduced the first such public two-party key exchange algorithm, which we call DH. There are other two-party key exchange protocols with cryptographic strength comparable to DH. For the purposes of this paper we will use DH as a place holder for any of these two-party protocols. As we first introduced with EGK [28], we can use any of these two-party protocols at any point in CCEGK, changing protocols for each pair of groups communicating and even between rounds. The replaceability of these protocols is unique to the binary-tree-based group key protocols.

7.3.2 Diffie-Hellman Protocol with indistinguishable security

Let g, p and q be publicly known values, where p, q are large primes and where $p = 2q + 1$. g is a generator of the group $\mathbb{QR}_p$ that is a subgroup of $\mathbb{Z}_p^*$ and the order is $|\mathbb{QR}_p| = q$. Let Alice and Bob choose secret private values x_A and x_B from $\mathbb{Z}_q = \{0, ..., q-1\}$, respectively. From these secret private values, they create their corresponding public values (**blinded keys**), r_A and r_B, and send them to each other.

$$\begin{aligned} Alice \rightarrow Bob &: r_A = g^{x_A} \, mod \, p \\ Bob \rightarrow Alice &: r_B = g^{x_B} \, mod \, p \end{aligned}$$

Bob and Alice then compute a shared secret key using their partner's public value and their own private value:

$$K = (r_A)^{x_B} \, mod \, p = (r_B)^{x_A} \, mod \, p = g^{x_A * x_B} \, mod \, p$$

An eavesdropper can know $p, q, g, g_a^x \, mod \, p, g_b^x \, mod \, p$ and still not be able to compute the shared key K. Furthermore, this Diffie-Hellman protocol is based on the **Decisional Diffie-Hellman** assumption. An eavesdropper can not distinguish between random keys in $\mathbb{QR}_p$ and keys that were generated by the above algorithm.

7.3.3 Group Key Protocols

The class of group key protocols we examine are those based on the utilization of underlying two-party key exchange algorithms such as the DH algorithm described above. The idea behinbdthese protocols is that the group is arranged in a binary tree hierarchy. Although other structures exist, the binary tree hierarchy has provided for the most efficient algorithms to date. At the base of the tree are pairs of nodes, each representing a group of size one. During the algorithm, two groups pair up and communicate their blinded keys to one another, forming a single shared key. These groups then form a new group and use the shared key to generate a new secret group key which is used to generate the new blinded key for the next merging operation. This style of tree-based algorithm appeared in the literature in the early 2000s as different research groups concurrently focused on this topic. The first suggestion of a tree-based approach was by Becker and Willie [61], but they did not develop a full protocol around the concept. Protocols emerged later, including EGK [28], TGDH [142, 145], and STR [143, 144], all based on contributory group key management to improve the cost of communication and computation.

7.3.4 Terminology

Group Key Management Operations

Throughout this paper we will address several group key management operations used by group key systems. The operations we describe are the following:

Initialization Operation This is the initial creation of the group key and organization of the key management infrastructure.

Join This operation brings a new member into the existing group.

Mass join (Mass add) This operation allows many new members to be added to an existing group simultaneously when these new members have not already formed a group of their own.

Merge (Group fusion) This operation, as opposed to mass join, is used when another group is combined with the existing group to become a new group.

Leave This operation is used to remove a member from the group.

Mass leave This operation is used when multiple members are simultaneously removed from the existing group.

Split (Partition, or group fission) This operation, different from mass leave, occurs when a single group is divided into two or more component groups.

Key refresh To prevent the secret key from becoming stale, it should be changed. Moreover, to prevent an adversary from breaking in, we should refresh the original key and generate a new secret key periodically.

Secrecy Terms

In the cryptographic protocol literature, there are several goals related to the management of group keys with respect to membership changes in the group. These goals are:

Group key secrecy Any generated group key is indistinguishable in polynomial time from a random number. This is called the "Decision Diffie-Hellman Problem" (DDH) [68]. For the protocols discussed in this paper, the cryptographic strength of the generated group key is solely dependent upon the strength of the underlying two-party protocols. That said, we will not address the DDH problem further in this paper.

Weak backward secrecy This ensures that new members cannot decrypt and understand previous messages that were sent prior to their joining the group.

Backward secrecy This ensures that a passive adversary who knows a contiguous subset of group keys can not discover preceding group keys.

Weak forward secrecy This ensures that the leaving members can no longer decrypt and understand messages after they have left the group.

Forward secrecy This ensures that a passive adversary who knows a contiguous subset of previous used group keys can not discover subsequent group keys.

Perfect forward secrecy This is a more strict version of the above. The active or passive attacker can additionally have a long-term secret key or session key of a current or past group member and still not compromise the communication.

n	number of group members
N	number of merging, joining or leaving members or subgroups
r	denotes a random integer
$< l, i >$	the i^{th} node at level l in the binary key tree
$K_{<l,i>}$	the secret key of the node $< l, i >$
$bK_{<l,i>}$	blinded key of the node $< l, i >$, e.g., $\alpha^{K_{<l,i>}} \ mod \ q$
bK_i	blinded key of the i^{th} group member.
M_i	the i^{th} group member, $i \in [1, n]$
K_i	secret key of the member M_i
α	an exponentiation base, which is a prime number based on DH
q	order of the algebraic group (large prime number for DH)
BT	the binary key tree

Table 7.2: Symbols used in metrics

Key independence This requires that anyone who knows a true subset of the group keys, or the private keys corresponding to a true subset of the blinded keys, is unable to guess an additional group key.

Evaluation Metrics

In our evaluation of the performance of the algorithms we evaluate many costs. This includes communication and computation costs for the evaluated operations. The values is these costs are depicted in Table 7.2.

Number of rounds This is a generic time unit used to compare the number of steps taken in different operations. The algorithms often require synchronization between rounds; therefore, this number becomes important when taking synchronization time into account.

Number of unicast messages This is the sum of the messages sent by each member to another single member in the group in the operation. This number is useful for determining total communication and is important if many or all nodes are on the same network collision domain, thus forcing these messages to be sent sequentially and not in parallel.

Number of broadcast messages This is the sum of the messages sent by each member to all the other members in the group for the operation. Since the messages go to all members of the group, it greatly affects total communication costs depending upon the underlying network topology[1].

Number of messages This is the sum of all unicast messages and broadcast messages. We use this number to determine the total time of communication in an underlying broadcast network.

Number of sequential exponentiations During an operation there will be a series of computationally expensive cryptographic operations (such as modular exponentiation used in the DH algorithm). The algorithms in the literature often require the results of one cryptographic operation prior to the execution of another. This metric represented the worst case scenario, the longest sequence of dependencies of these costly cryptographic operations in the operation.

Number of signatures This is the sum of digital signatures used in every round. In every round the sponsor sends one digital signature.

[1]The existing group key literature assumes a fully connected broadcast network.

Number of verifications Given that each message needs to be verified, the number of verifications is equal to the number of messages; however, several verifications can occur in parallel, so we need to concern ourselves with the number of sequential verifications – the maximum number of verifications that must occur during an operation.

In addition, we define the *sponsor* as the group member elected to coordinate group activities. In our prior work, we have used the term *group controller* for this designation; however, sponsor seems to be a more fitting appellation.

7.4 CCEGK Protocol

In this section, we discuss our communication-computation efficient group key (CCEGK) algorithm. This algorithm is based on EGK, which we presented a few years ago [28], and adopts conventions and a few operations from TGDH [142, 145]. EGK was originally created with the concept of rapid addition of new members and low rekeying costs. TGDH and STR adopt a more balanced approach but also use tree-based group keying. CCEGK is an attempt to merge these two concepts.

Having extended EGK and TGDH, CCEGK is based on a binary tree structure that assumes a consistent world view among all group members. Specifically, we make the following assumptions which are consistent with those made in the group-key literature:

1. All members know the key tree structure, an ordered list of each other's identities, and their initial position in the tree.
2. All participants can unambiguously determine their group sponsor.
3. Every member sees the same sequence of group key operations.

To implement a consistent world view, TGDH and STR use View Synchrony (VS), a simple specification that allows processes to synchronize on specific views [106]. According to the authors of TGDH and STR, VS is essential for any fault tolerant group key agreement protocol. Therefore, as with TGDH and STR, we assume an implementation that supports the semantics of VS.

These assumptions are straightforward but have sweeping ramifications that are necessary to facilitate current group key exchange algorithms. It is understandable that the current group key exchange algorithms would make use of these assumptions, as the cost of communication for a traditional wired network is fairly small. However, less traditional networks, including wireless, will need a reassessment of these assumptions; we leave that discussion to a future paper.

Given these assumptions, a set of nine operations emerge from the group key field. Almost every proposed protocol is presented with split, merge, add, leave, mass add, and mass leave. However, initialize, rebalance, and key refresh are potential exceptions, as we indicate below. In this section we define each of these nine operations as they are implemented in CCEGK. We also provide the CCEGK communication and computation costs for each operation. For completeness we include digital signature and verification costs if they are implemented in the protocols. We leave discussion of signatures and signature verification to Section 7.6. In Sections 7.7 we provide a theoretical comparisons of the CCEGK costs to those of EGK, STR and TGDH.

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential Exponentiations	Signatures	Verifications
h	$2n-2$	n	$n-2$	2h-2	h	$2n-2$

Table 7.3: Costs of initialization operation

Throughout the remainder of this section, we chose to use terminology and conventions established in the TGDH papers since they have been published in venues with a wider distribution than the original EGK paper.

7.4.1 Initialization Operation

The **initialization** operation is the group genesis algorithm. In this algorithm each node pairs up with its neighbor, if available, to establish a new shared key. If no neighbor is available (as in an incomplete binary tree), the single node will behave as an atomic group. The new group then repeats the pairing with other groups until there is only one group. The basic operation occurs as follows:

1. Suppose that there is a collection of entities: $\{ M_1,\ldots, M_n\}$ (it is not important whether n is equal to 2^r). From assumption 1, we know all members can sort their identities in some order. Each member represents the sorted set of identities as the labels for the sponsors of a binary tree.

   (a) In round 1, each node M_i generates a DH key pair consisting of a secret key K_i and a corresponding blinded key $bK_i = \alpha^{K_i}\ mod\ p$. Each node M_i unicasts a message including its blinded key to its sibling. M_i performs a two-party Diffie Hellman (DH)[2] with its sibling to calculate a new DH key pair with the private key and the blinded key. Each node has a single parent and single sibling. We choose the rightmost node as the sponsor of the group (if n is not even, the last node does nothing in the first round; we treat it as a group. The same condition is in every round). We treat every new group as a new node, so the new number of nodes is $\lceil \frac{n}{2} \rceil$.

   (b) In subsequent rounds, each member of the group determines the new group private and blinded keys. The sponsor of each group broadcasts a message including its group's new blinded key to all of the members of its sibling group. Every member then performs a two-party DH with the blinded key of its sibling group to generate a new group with a new DH key pair. We again choose the rightmost node as the sponsor of this new group.

   (c) We repeat the above process until round $h = \lceil \log_2 n \rceil$($h$ is the height of the key tree). At this point we have a single group, G, which includes all the members, each sharing the group secret key $K_{<0,0>}$.

The costs of the CCEGK initialization operation are summarized in Table 7.3.

[2]Recall that although we use DH in this paper, any equivalent two-party public key algorithm can be used in place of DH. The choice of algorithm can be based on the pair of nodes involved in the exchange.

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential Exponentiations	Signatures	Verifications
1	2	1	1	1	1	2

Table 7.4: Costs of the Join Operation

7.4.2 Join Operation

The **join** operation occurs when a new joining member sends a join request to the group. The procedure for the join operation is as follows:

1. Suppose that the group has n members: $\{M_1, \ldots, M_n\}$. The new member, M_{n+1}, broadcasts a message including its own blinded key to every member of the group. Simultaneously the sponsor of the original group (the shallowest rightmost leaf) unicasts the group's blinded key to M_{n+1}.
2. The group and the new node create a new group. The new root key node has two children: the root node of the original tree on the left and M_{n+1} on the right. Every member can now compute the group key because all original members only need the new member's blinded key, and the new member M_{n+1} needs only the blinded group key of the original group.

The costs of the join operation are summarized in Table 7.4. We assume that before the join operation, the sponsor of the original group and a new joining member generate their own blinded key and therefore do not include these costs in the table.

CCEGK, as do EGK and STR, always joins at the root of the tree, resulting in potentially far fewer sequential exponentiations. TGDH will join at the root only if the tree is a full tree, in an attempt to keep the tree more balanced. STR ,on the other hand always has a skinny tree, where every internal node had one child that is a leaf.

For purposes of comparison, the left-hand side of Figure 7.1 shows a sample tree taken from Figure 2 of the TGDH paper [145]. In this example, member M_4 is joining the group $BT1$. In [145] the new node is added to the bottom of the tree. In this example, the sponsor M_3 performs the following actions:

1. generates a new root node and a new member node.
2. sets the new root node as the parent node of the original root node and the new member node, denoted as node $< 0, 0 >$.

After these actions, all members in the original group know $bK_{<1,1>}$; and the new member M_4 knows $bK_{<1,0>}$, and if it has previously computed its own blinded key, then every member can compute the new group key in one round with one sequential exponentiation.

7.4.3 Merge Operation

The **merge** operation occurs when two or more subgroups wish to merge into one group. The sponsor announces the merge event for these subgroups. These subgroups receive the notification

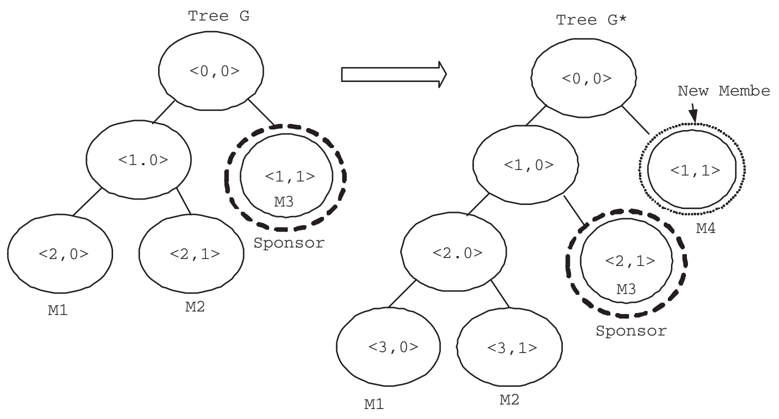

Figure 7.1: Tree updates from the join operation.

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential Exponentiations	Signatures	Verifications
1	N	0	N	N	1	N

Table 7.5: Costs of Merge Operation

and they perform DH exchanges to establish a new group key. The procedure of the two-subgroup merge operation is as follows:

1. Suppose we have two groups G_1 and G_2 in the first round. The sponsor of each subgroup (the shallowest rightmost leaf) broadcasts a merge request message including the blinded key of the group to the members of the other group.

2. G_1 and G_2 create a new group. The new root key node has two children: the root node of G_1 on the left and the root node of G_2 on the right. Every member in the new group can compute the group key because all the members in G_1 know the blinded group key of G_2 and all the members in G_2 know the blinded group key of G_1.

Figure 7.2 shows an example of Group $G_1 = \{M_1, M_2, M_3\}$ merging with Group $G_2 = \{M_4, M_5\}$, where the sponsor is M_3. Since all members in G_1 know $bK_{<1,1>}$, and all members in G_2 know $bK_{<1,1>}$, every member can compute the group key in one round. The costs of the two-subgroup merge operation are summarized in Table 7.5.

We can extend the merge operation to an N-subgroup merge operation: the sponsor of every subgroup broadcasts the blind key of its own group key to every member, including the group members and the members of the other subgroups. Every member knows his position in the key tree. The sponsor of the new group internally determines the iterative merger of the group as above, calculating the blinded keys up the path to the root. It then broadcasts these to all members who can then internally rekey. So in an N-subgroup merge operation we only need one round, N messages, and N sequential exponentiations.

CCEGK, as with EGK, merges the two roots of the trees, resulting in lower cost but a potentially more unbalanced tree. TGDH merges the shallower tree to the deeper one, or at the root, whichever creates the shallowest final tree. STR adds one tree to the bottom of the other.

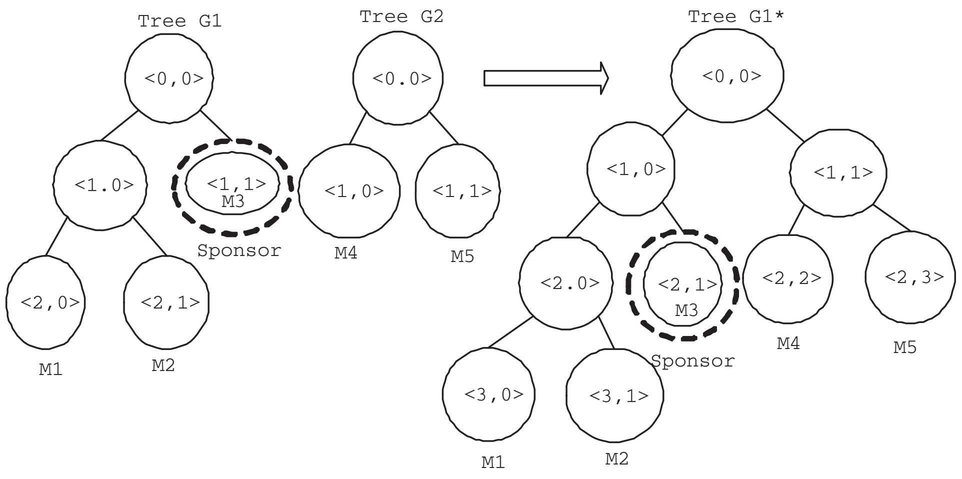

Figure 7.2: Tree updates from the merge operation.

7.4.4 Mass Join (N-Member Join) Operation

The **mass join** operation occurs when two or more joining members send join requests to the group sponsor. The sponsor then announces the mass join event to the current group and the joining members. The current group and these members receive the notification and they perform DH exchanges to establish a new group key. EGK [28] put forward two ways to implement this mass join. In this paper we still use these two methods and supplement them with a new method.

1. Mass Join-Iterative [28]: treats N members as N separate single member joins.
2. Mass Join-Merge [28]: initialize the N entities to form their own independent group, then merge this group with the original group to generate a new group.
3. Mass Join-Simultaneous: The sponsor of the current group and N joining members broadcast a message including their blinded key to every member. Due to VS we know that every member has received this set of blinded keys before starting the mass-join operation. The sponsor then calculates all the blinded keys created as if these members were being added sequentially and broadcasts the keys to all members. Each member can determine the same view of the system and can calculate his position in the key tree; every member can generate the group key of the new group. In this method, we use one round, $N + 1$ messages, and $2(N - 1)$ sequential exponentiations.

The costs of the mass join operation are summarized in Tables 7.6-7.7.

Mass Join-Iterative is most expensive among these three methods in the communication and computation costs. In communication costs, Mass Join-Simultaneous is more efficient than Mass Join-Merge, whereas in computation costs, Mass Join-Merge is more efficient than Mass Join-Simultaneous. In this paper, we use the Mass Join-Simultaneous method when we compare the costs of communication and computation in the mass join operations of the other protocols.

Method	Communication			
	Rounds	Messages	Unicast	Broadcast
Mass Join-Iterative	N	$2N$	N	N
Mass Join-Merge	$h' + 1$	$2N$	N	N
Mass Join-Simultaneous	1	$N + 1$	0	$N + 1$

Table 7.6: Communication Cost of Mass Join

** h' is the height of the key tree that is created by N joining members, $h' = \lceil \log_2 N \rceil$

Method	Computation		
	Sequential Exponentiations	Signatures	Verifications
Mass Join-Iterative	$2N - 1$	N	$2N$
Mass Join-Merge	$2h'$	$h' + 1$	$2N$
Mass Join-Simultaneous	$2(N - 1)$	1	$N + 1$

Table 7.7: Computation Cost of Mass Join

** h' is the height of the key tree that is created by N joining members, $h' = \lceil \log_2 N \rceil$

7.4.5 Leave Operation

The **leave** operation occurs when a group member sends a leave request to the group. The sponsor announces the leave event to the remaining group members, who then do DH exchanges to create a new shared key. In EGK we required a leave operation to fully reinitialize the tree. To reduce costs, the procedure of the CCEGK leave operation is very similar to that in TGDH [142, 145]:

1. Suppose that we have n members in the group, and a group member M_i wants to leave the group. The sponsor is the shallowest rightmost sponsor of the subtree rooted at M_i's sibling node. In the leave operation, every member updates its key tree by deleting the sponsor corresponding to M_i. The former sibling of M_i replaces M_i's parent node. The sponsor picks a new secret key and computes all keys on its key path up to the root.

2. After the sponsor generates all the keys, it broadcasts a message including all the blinded keys to the other members in the group. After the other group members receive the message, they can compute the group key for each node on up the new tree.

Figure 7.3 shows an example of member M_3 leaving from a group where the sponsor is M_2.

The costs of the leave operation in the worst case are shown in Table 7.8. The total sequential exponentiations in the leave operation depend on the position of the leaving member. For purposes of evaluation, we will only consider the worst case in the leave operation.

Since our join and merge operations add to the root node, our tree can become very unbalanced. At some point the cost of the above leave operation may be greater than the cost of initializing a new group. Under such cases, instead of the above operation, we will invoke a rebalance operation (see Section 7.5).

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential Exponentiations	Signatures	Verifications
1	1	0	1	3h-3	1	1

Table 7.8: Costs of Leave Operation

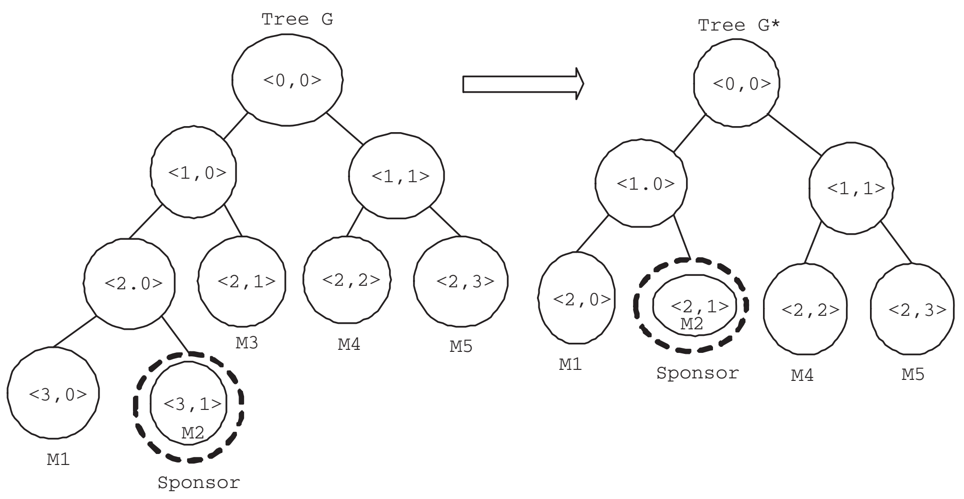

Figure 7.3: Tree updates from the leave operation.

7.4.6 Mass Leave (N-member leave) Operation

The **mass leave** operation occurs when two or more group members send leave requests to the group. The sponsor announces the mass leave event to the remaining group members. The remaining group members receive this notification and do DH exchanges to create a new group key. There are two ways to implement the mass leave operation, depending on the number of leaving group members. Suppose that we have n members in the group and N members will leave the group. The two methods are as follows:

(I) **Mass Leave-Balanced**: When the number of leaving group members is very large, it is more efficient to reconstruct the key tree using the initialization operation in Section 7.4.1. The total number of rounds is $\lceil \log_2(n-N) \rceil$; the total number of messages is $2(n-N-1)$. The sequential exponentiations are $2\lceil \log_2(n-N) \rceil$. In this method, the updating key tree becomes balanced. Consequently, a later rebalance scheme is not needed.

(II) **Mass Leave-imbalanced**: When there are fewer leaving group members, it is more efficient to use the Mass Leave-Imbalanced method. The procedure is as follows:

1. The sibling of every leaving node replaces its parent node. Every remaining member updates its key tree by deleting all leaving members. For every leaving node, we determine its sibling node and its parent node and replace its parent node with its sibling node. Then every sponsor, after deleting every leaving node, uses the same criteria as described in the leave operation. We choose the shallowest rightmost sponsor as the sponsor in the new key tree. To prevent reusing the old group key, the shallowest rightmost sponsor in the array picks a new secret key. We update the order number and the level of every node in the new tree. We put every sponsor into two initial empty arrays: A, called sponsor array, and B, called intermediate node array.
2. The elements in A calculate their secret keys and blinded keys on their key-path as far as possible in parallel. For example, an element $< k, m >$ in B can calculate its secret keys and blinded keys up to the node $< l, i >$. We check whether $< l, i >$ is already in B. If

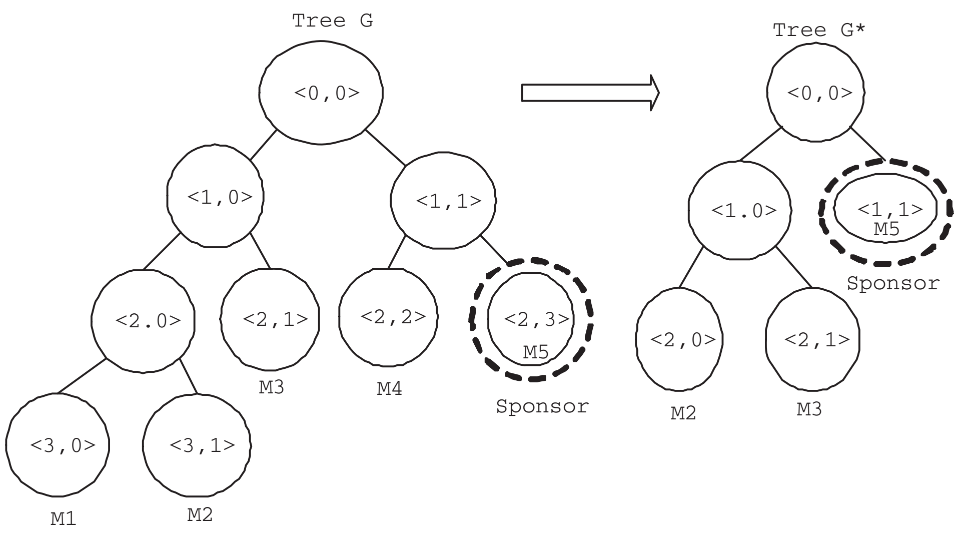

Figure 7.4: Tree updates from the mass-leave operation.

Table 7.9: Communication Cost of Mass Leave

Method	Communication			
	Rounds	Messages	Unicast	Broadcast
Balanced	h	$2(n-N-1)$	$n-N$	$n-N-2$
Imbalanced	$\min(\lceil \log_2 N \rceil + 1, h')$	$\min(2N, \lceil n-N \rceil)$	0	$\min(2N, \lceil n-N \rceil)$

not, we insert the node $< l, i >$ into B, and remove $< k, m >$ from B. And so on for the other elements in B. After that, we check whether elements in B have an offspring-ancestor relationship. If they do, we will remove all the offspring from B. Now, for every element $< l, i >$ in B, we will search the nodes in A which are the offspring of $< l, i >$, keep the shallowest rightmost node in A and remove the other offspring from A. Now every element in A will broadcast all of its blinded keys to all the members in the new balanced tree. After receiving the messages, all the members calculate the blinded keys and secret keys on their key-path as far as possible in parallel.

3. We iterate Step 2 until the number of elements in B is 1 and the element is the root node.

In the second method, the updating key tree is still imbalanced. Because rounds, messages and sequential exponentiations in the mass leave operation depend on the position of the leaving members, we will consider the worst case in the mass leave operation. In the worst case, we need $\min(\lceil \log_2 N \rceil + 1, h)$ rounds, $\min(2N, n-N)$ messages and $3h-3$ sequential exponentiations (note that in general h is larger than $\lceil \log_2(n-N) \rceil$). Figure 7.4 shows an example of members M_1 and M_4 leaving from a group using the second method.

The worst-case costs of the mass leave operation are summarized in Tables 7.9-7.10.

From Tables 7.9 and 7.10, we know the number of messages and verifications in Mass Leave-Balanced are greater than those in Mass Leave-Imbalanced. Total sequential exponentiation in Mass Leave-Balanced are smaller than that in Mass Leave-Imbalanced, in both the average and worst case. When $N \geq \frac{n}{2}$, the number of rounds and signatures in Mass Leave-Balanced are smaller

Method	Computation		
	Sequential Exponentiations	Signatures	Verifications
Balanced	$2h$	h	$2(n-N-1)$
Imbalanced	$3h'-3$	$\min(\lceil\log_2 N\rceil+1, h')$	$\min(2N, \lceil n-N\rceil)$

Table 7.10: Computation Cost of Mass Leave

** h and h' are the height of updating key tree after N members are leaving from the group using the Mass Leave-Balanced and Mass Leave-Imbalanced schemes respectively. $h = \lceil\log_2(n-N)\rceil$ and in general h' is larger than h.

Method	Balanced	Imbalanced
Rounds	$\max(h_i)$	$\max(\min(\lceil\log_2 P_i\rceil+1, h_i{}'))$
Messages	$max(2(n-P_i-1))$	$max(min(2P_i, n-P_i)$
Unicast	$max(n-P_i)$	0
Broadcast	$max(n-P_i-2)$	$max(min(2P_i, n-P_i)$

Table 7.11: Communication Cost of Partition

than those in Mass Leave-Imbalanced. Otherwise, the number of rounds and signatures in Mass Leave-Balanced are larger than those in Mass Leave-Imbalanced.

7.4.7 Partition Operation

The **partition** operation occurs when the current group will split into two or more subgroups. The sponsor announces the partition event to every subgroup. Every subgroup receives this notification and performs DH exchanges to create a new group key. The partition operation is similar to the mass leave operation; it can apply two different methods: **Partition-Balanced** and **Partition-Imbalanced** to the partition operation corresponding to Mass-Leave-Balanced and Mass-Leave-Imbalanced methods. Suppose that we have n members in the group and the group will be split into N separate subgroups $G_1, G_2, \ldots G_N$. For every subgroup, we treat the group members of the other subgroups as leaving members. Therefore, the partition operation can be treated as an N-mass leave operation. Note that the N-mass leave operations are simultaneous. The total number of rounds is the largest of the rounds from the separate N-mass leave operations, and the number of sequential exponentiations is the maximum sequential exponentiations among the N-mass leave operations. The number of messages is the largest of all the messages in every mass leave operation. Figures 7.4 and 7.5 are examples of splitting two subgroups from the original group. The first of these figures is identical to that of the mass-leave operation. The two subgroups are $G_1\{M_2,M_3,M_5\}$ and $G_2\{M_1,M_4\}$, respectively.

Suppose the total size of leaving members for subgroup G_i is P_i. h_i and $h_i{}'$ are the height of updating key tree in the ith subgroup ($1 \le i \le N$) in Partition-Balanced and Partition-Imbalanced, respectively (note that in general $h_i{}' > h_i$). The worst case costs of the Partition operation are summarized in Tables 7.11-7.12. Since this is a partition operation and not a mass leave, we consider the costs of reforming both subgroups in contrast to the costs published by Kim et al. [142, 145] which only look at the costs of one of the subgroups.

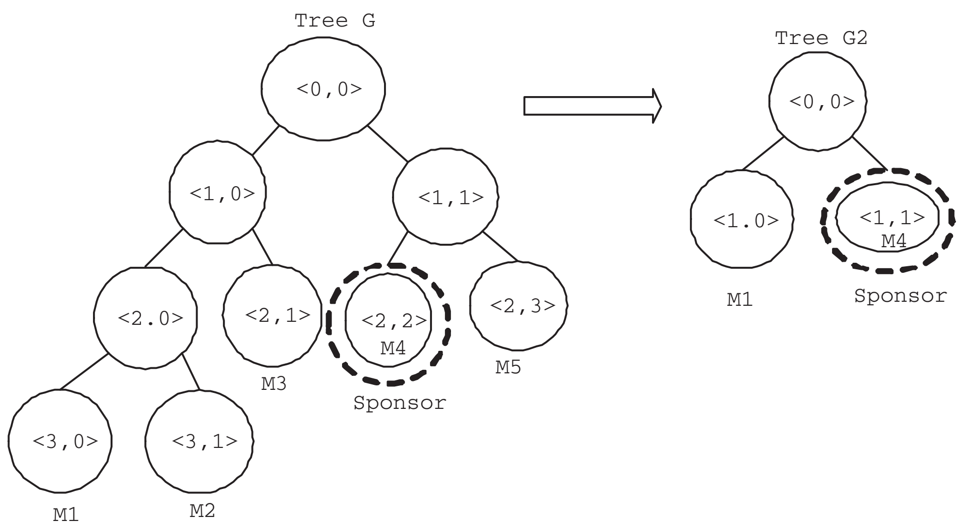

Figure 7.5: Tree updates for subgroup G_2 in the partition operation.

Method	Computation		
	Sequential Exps.	Signatures	Verifications
Balanced	$\max(2h_i)$	$\max(h_i)$	$\max(hi)$
Imbalanced	$\max(3{h_i}') - 3$	$\max(\min(\lceil \log_2 P_i \rceil + 1, {h_i}'))$	$\max(min(2P_i, n - P_i))$

Table 7.12: Computation Cost of Partition

7.4.8 Key Refresh

To avoid exposure of the group's secret key, we should refresh it periodically. The group's sponsor picks a new secret key and computes its keys and blinded keys up to the root of the tree. After computing the group key, it broadcasts a message including all the blinded keys to the other members. Upon receiving the message, the other members can calculate their keys and blinded keys.

Because sequential exponentiations depend on the position of the group sponsor, we will consider the worst case costs in the key refresh operation, summarized in Table 7.13.

7.5 Rebalance Scheme

After numerous add, merge, leave, mass leave, and partition operations, the key tree can become quite unbalanced. So, when the key tree reaches a certain imbalance point, usually caused by leave, mass leave, or partition operations, we should rebalance the tree and treat the rebalance

Communication				Computation		
Rounds	Messages	Unicast	Broadcast	Sequential Exponentiations	Signatures	Verifications
1	1	0	1	3h-3	1	1

Table 7.13: Costs of Key Refresh

operation as a leave, mass leave, or partition operation. The imbalance point can vary depending on network and efficiency requirements. One possible rule of thumb is to rebalance during a leave operation when the cost of rebalancing exceeds that of leaving. We discuss two ways to implement the rebalance scheme on a tree of size n:

1. Choose a subtree as the basic tree. To rebalance we have two requirements, the height, h, of the basic tree is $\lceil \log_2 n \rceil$ and the basic tree is more densely populated than the other subtrees of the same height. The height of the basic tree is easy to calculate, and it is also easy to determine which subtree is the most dense by counting the number of members. After we determine the basic tree, we choose the shallowest rightmost sponsor in the basic tree as the sponsor in the new balanced tree. We treat every group member which is not in the basic tree as a separate entity and insert them into the basic tree from the shallowest level down to the deepest level. After we finish inserting all the entities, we update the order number and the level of every node in the new tree. The height of the new key tree is h and the new key tree becomes more balanced, if $\log_2 n$ is an integer, the tree is a full binary tree.
2. Alternatively, we divide the original tree into several subtrees. The height of every subtree is at most h. The division procedure is as follows:
   (a) We choose a subtree with the largest number of members and height no more than h. We split off that subtree.
   (b) We repeat Step (a) to split every remaining subtree, until the height of every remaining subtree is no more than h. Note that in the end the heights of some subtrees may be less than h.
   (c) We sort these subtrees from highest height to lowest height and from largest to smallest group size. We denote the largest subtree as the basic tree. We treat every subtree as an entity. We choose the shallowest rightmost node of every entity as the sponsor of the entity. The sponsor of the basic tree determines the insertion position for every entity and informs the other subtree sponsors of the positions in the basic tree. Our insertion strategy is as follows: we insert the subtrees sorted from highest to lowest height into the nodes from the shallowest level down to the deepest level. If the joining entities increase the height of the basic tree, they join to the root node of the basic tree. After we finish inserting all the entities, we update the order number and the level of every node in the new tree.

Picking one of the above methods, we perform the following actions:

1. We put every sponsor into two initial empty arrays: A, the sponsor array, and B, the intermediate node array. The nodes are sorted by depth order. The elements in A calculate their secret keys and blinded keys on their key-path as far as possible in parallel. For example, an element $< k, m >$ in B can calculate its secret keys and blinded keys up to the node $< l, i >$. We check whether the node $< l, i >$ is already in B; if not, we insert the node $< l, i >$ into B. We then remove $< k, m >$ from B, and so on for the other elements in B. After that, we check whether some elements in B have an offspring-ancestor relationship. If so, we will remove all the offspring from B. Now for every element $< l, i >$ in B, we will search the nodes in A whic are the offspring of $< l, i >$, keep the shallowest rightmost node, and remove

the other offspring from A. Now every element in A broadcasts all its blinded keys to all the members in the new balanced tree. After receiving the messages, all the members calculate the blinded keys and secret key on their key-path as far as possible in parallel.

2. We iterate Step 1 until the number of elements in B is one and that element is the root node.

Suppose the number of total entities including the basic tree is N, and (using the second method) the height of the new rebalanced tree is h'. In the first method, in the worst case, we need $\min(\lceil \log_2 N \rceil + 1, h)$ rounds, $2N$ messages, and $3h - 3$ sequential exponentiations. The advantage of the first way is that it can let the tree become a full binary tree, and the number of rounds and sequential exponentiations are very small. The drawback, however, is that N may be very large because we treat every group member that initially is not in the basic tree as an entity and they could be quite numerous. The total communication cost is quite expensive when N is very large. In the second method, in the worst case, we need $\min(\lceil \log_2 N \rceil + 1, h')$rounds, $\min(2N, n - N)$ messages, and $3h' - 3$ sequential exponentiations. The advantage of the second method is that N is much smaller than that of the first method. Although the height of the key tree might be increasing, it increases slowly. Therefore, the number of the rounds and sequential exponentiations are still small, but the number of messages is much smaller than that of the first method. The drawback is that the new tree is still unbalanced. So, when N is very small, we shall use the first method. If N becomes very large, we will consider using the second method. Note that in EGK, the key tree will be rebalanced in the leave, mass leave, and partition operations by reconstructing the new key tree. Although the number of rounds is h, the communication and computation costs are very expensive because the number of rounds is $2n$, as is the number of verifications. In TGDH after the merge, leave, mass leave and partition operations, the key tree also becomes unbalanced but they do not provide a rebalance scheme, rather they rely on the heuristic choices of insertion points for joins and merges to maintain a partial balance. In STR, the key tree is the most unbalanced compared to EGK, TGDH, and CCEGK, and STR does not provide a rebalance scheme.

Figure 7.6 shows an example of the first rebalance scheme. We perform the following actions:

1. We will choose the subtree whose root node is $< 2, 0 >$ as a basic tree and M_4 as the group controller. Suppose at first, the sponsor array A and the intermediate node array B are empty. Node $< 2, 1 >$ is the sibling of $< 4, 1 >$, and $< 1, 1 >$ is the sibling of $< 3, 1 >$. The key tree is updated where $< 5, 0 >$ becomes $< 3, 0 >$ in the new tree, and so on. We put $< 3, 3 >$, $< 2, 2 >$, and $< 2, 3 >$ into A and B.

2. In the first round, $< 3, 3 >$, $< 2, 2 >$ and $< 2, 3 >$ calculate their blinded keys and secret keys. Node $< 3, 3 >$ can calculate the secret keys $K_{<2,1>}$ and $K_{<1,0>}$ and blinded keys $bK_{<2,1>}$ and $bK_{<1,0>}$ because it knows the blinded keys $bK_{<3,2>}$ and $bK_{<2,0>}$. We put $< 1, 0 >$ into B and remove $< 3, 3 >$. Node $< 2, 2 >$ can calculate the $K_{<1,1>}$ and $bK_{<1,1>}$ because it knows $bK_{<2,3>}$, we remove $< 2, 2 >$ from B and insert $< 1, 1 >$ into B. We remove $< 2, 3 >$ from B, because in A, $< 2, 2 >$ and $< 2, 3 >$ are the offspring of $< 1, 1 >$ in B. We keep $< 2, 2 >$ in A and remove $< 2, 3 >$ from A, then $< 3, 3 >$ and $< 2, 2 >$ broadcast their message to every member. Every member can calculate its blinded keys and secret keys.

3. We remove $< 1, 0 >$ and $< 1, 1 >$ from B and insert $< 0, 0 >$ into B. We remove $< 3, 3 >$ from A. Node $< 2, 2 >$ broadcasts all the blinded keys to every member, every member can calculate the group key.

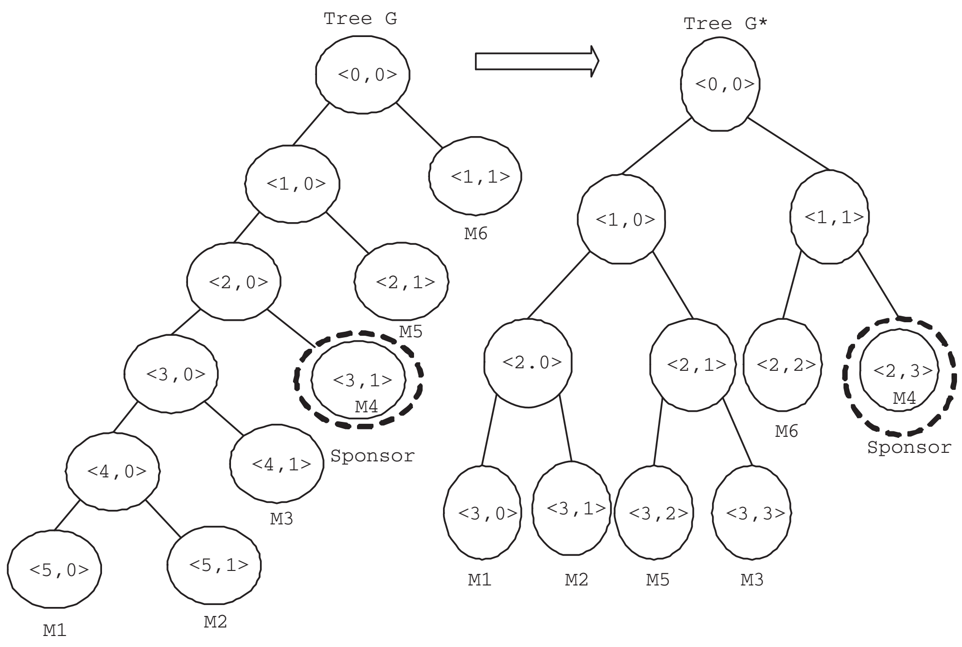

Figure 7.6: Tree operations from the rebalance scheme.

7.6 Authentication

Although the algorithm of CCEGK based on DH is secure and not easy to break, the entire system is vulnerable if the keys are not securely distributed. Therefore, we should implement an authentication algorithm in CCEGK. Our protocol will authenticate any operation, such as an addition of new members, a leaving group member, or mass leaving group members. We suppose that it is acceptable for this authentication to use public-key authentication. There are several ways to authenticate a group key exchange, such as centralized authentication, implicit authentication, and pairwise authentication [28, 54, 71, 138, 212]. The authentication of CCEGK can use any of the above authentication schemes. Centralized authentication is based on a digital signature and a public key infrastructure. A group may assign a single trust sponsor to perform authentication for the whole group, which will occur in the first step of the key exchange algorithm. Every group member sends its public key PK_i using an authenticated message. The sponsor validates the authenticity of the senders and sends a message to all authenticated group members. The drawback of centralized authentication is that the certificates are to be exchanged, so they consume bandwidth. Implicit authentication authenticates the sponsor and trusts the sponsor to have authenticated group members. This method directly maps to authenticated two-party key exchange algorithms in which the group members authenticate the blinded key transmitted by their partner. In pairwise authentication, every member encrypts a message with the group key and then broadcasts it in an authenticated message to all group members. If any of the authentications fails, the group key is discarded and a new key is generated without the unauthenticated member. In this paper, in order to compare the computation cost between the four protocols, for convenience and because TGDH and STR use the centralized authentication scheme, we suppose CCEGK uses the centralized authentication scheme (EGK supports centralized, implicit, and pairwise authentication and gives the algorithms in detail in [28]).

Protocols		Communication			
		Rounds	Messages	Unicast	Broadcast
CCEGK	Initialization	h	$2n-2$	n	$n-2$
	Join	1	2	1	1
	Mass Join	1	$N+1$	0	$N+1$
	Merge	1	N	0	N
	Leave	1	1	0	1
	Mass leave	$\min(h'+1, h)$	$\min(2N, n-N)$	0	$\min(2N, n-N)$
EGK	Initialization	h	$2n-2$	0	$2n-2$
	Join	1	2	0	2
	Mass Join	$h'+1$	$2N$	0	$2N$
	Merge	N	$2N-2$	0	$2N-2$
	Leave	h	$2(n-1)$	0	$2(n-1)$
	Mass leave	h	$2(n-N)$	0	$2(n-N)$
TGDH	Initialization	h	$2n-2$	0	$2n-2$
	Join	2	3	0	3
	Mass Join	$h'+1$	$2N$	0	$2N$
	Merge	$h'+1$	$2N$	0	$2N$
	Leave	1	1	0	1
	Mass leave	$\min(h'+1, h)$	$\min(2N, n-N)$	0	$\min(2N, n-N)$
STR	Initialization	$n-1$	$2n-2$	0	$2n-2$
	Join	2	3	0	3
	Mass Join	2	$N+2$	0	$N+2$
	Merge	2	$N+1$	0	$N+1$
	Leave	1	1	0	1
	Mass leave	1	1	0	1

Table 7.14: Table in Comparison of Communication

7.7 Theoretical Comparison of Four Protocols

In this section, we analyze the complexity of the four protocols CCEGK, EGK, TGDH, STR. Complexity includes the cost of communication (such as number of rounds, number of messages, number of direct messages, number of broadcast messages) and the cost of computation (number of sequential exponentiations, signatures, and verifications).

Because communication cost is important in high-delay networks, we cannot ignore it. A protocol is needed to balance the cost. Tables 7.14 and 7.15 summarize the costs of communication and computation for every operation among four protocols in the worst case situation.

The number of current group members, mass join members, and mass leave members is n; the number of merging groups (including the original group) is N and the number of merging members is m. We use h to denote the height of the updating key tree and and define h' as $\lceil \log_2 N \rceil$. It is noted that Kim et al. do not implement the initialization operation in TGDH and STR. The total number of messages and total verifications in the mass leave operation of TGDH are inconsistent with each other in [145] and [38]. In this paper, we implement the initialization operation in TGDH and STR, and we use a modified value explained in the Appendix that seems to better fit their description of their algorithm,; see the Appendix for further details. It is important that we see how these protocols operate when multiple operations are executed. In [282] we therefore compare the average costs of multiple instances of these operations to get a better feel by experimental simulation.

Protocols		Computation		
		Sequential Exponentiation	Signatures	Verifications
CCEGK	Initialization	$2h-2$	h	h
	Join	1	1	1
	Mass Join	N	1	1
	Merge	$N-1$	1	1
	Leave	$3h-3$	1	1
	Mass leave	$3h-3$	$\min(h'+1,h)$	$\min(2N,n-N)$
EGK	Initialization	$2h-2$	h	h
	Join	1	1	2
	Mass Join	$2h'$	$h'+1$	$h'+1$
	Merge	$2N$	N	$2N-2$
	Leave	$2h$	h	h
	Mass leave	$2h$	h	h
TGDH	Initialization	$2h-2$	h	h
	Join	$3h-3$	2	3
	Mass Join	$3h-3$	$h'+1$	$2N$
	Merge	$3h-3$	$h'+1$	$2N$
	Leave	$3h-3$	1	1
	Mass leave	$3h-3$	$\min(h'+1,h)$	$\min(2N,n-N)$
STR	Initialization	$2(n-1)$	$n-1$	$n-1$
	Join	4	2	3
	Mass Join	$3N+1$	2	$N+2$
	Merge	$3m+1$	2	$N+1$
	Leave	$\frac{3n}{2}+2$	1	1
	Mass leave	$\frac{3n}{2}+2$	1	1

Table 7.15: Table in Comparison of Computation

Part II

Multiple Independent Levels of Security Architecture

Overview of Part II

The first two parts of this report discussed network authentication protocol design and analysis. Based on this work we found that we had to develop a new architecture to support execution of the protocols. This portion of the report summarizes our results in this area.

The work highlighted in this section has been published in several papers, and results in masters thesis and dissertations. The abstracts for these publications can be found in final section of this report. Publications related to design and analysis of MILS system are filed under the following keys: Alves-Foss02a [29], Alves-Foss04a [32], Alves-Foss04b [34], Alves-Foss06a [30], ConteDeLeon02b [81], ConteDeLeon06a [82], ConteDeLeon06b [83], ConteDeLeon07a [84], Dai03a [88], Hanebutte05a [124], Harrison05a [127], Joy06a [135], Oman04a [209], Robinson07a, [221], Robinson07b [222], Robinson07c [223], Rossebo06a [227], Son06a [246], Son07a [247], Son08a [245], Taylor04a [254], Wahsheh06a [261], Wahsheh07a [262], Wahsheh07b [263], Wahsheh08a [260], Wahsheh08b [264], Wahsheh08c [265], Wang06e [267], Wang06f [266], Yang05d [274], Zhou06c [287], and Zhou08a [285].

Specifically we provide a summary of the MILS architecture, and overview of our design guidance work in terms of traceabilty issues and security policy refinement. This design guidance helps in building secure systems that can be used to deploy trusted network authentication protocols.

Chapter 8

Multiple Independent Levels of Security Overview

8.1 Introduction

The design and implementation of secure applications is a daunting task. Especially since these applications rely upon the security of the underlying operating system and services. Too often, a secure service or application will be compromised by a security flaw in a supporting service or operating system. The *Multiple Independent Levels of Security and Safety (MILS)* approach remedies this situation by providing a reusable formal framework for high assurance system specification and verification.

High assurance systems are those that require convincing evidence that the system adequately addresses critical properties such as security and safety [131]. If the high assurance system fails to meet its critical requirements, then there is a potential for security breach or loss of life. Since such systems are so critical, there is a need for a rigorous design and analysis process. The avionics community has understood this for years and has developed a set of guidelines for the design, analysis and evaluation of safety systems [228, 229]. Although very rigorous and adequate for safety of airborne computing systems, it is insufficient to meet the security concerns of high-assurance security systems; systems that protect national security interests. The Common Criteria (CC) provides guidance for design, analysis and evaluation of security critical systems that includes a very high level of assurance [53]. At the higher levels of assurance, the CC requires the use of formal methods, mathematical models and proofs.

It is commonly believed that the successful deployment of a high assurance secure application requires the existence of an underlying secure operating system and services. Unfortunately, attempts to develop secure, general-purpose operating systems have failed to be supportable. They have failed for various reasons, but predominately due to the fact that they were architected to do too much and the requisite formal methods used became too difficult to manage. Historically, a high assurance operating system was based on the concept of a security kernel and Trusted Computing Base (TCB), all of which required formal methods evaluation. Two such systems that became unsupportable are Blacker [269] and Caneware [224]. Through the development lifecycle they included more and more functionality into the trusted computing base, requiring too much effort in the development of the supporting formal methods.

For embedded systems, the need for a secure operating system is no less critical. While user access is not a problem, secure communications and safe process execution is still a concern. Careful development and verification of the operating system and trusted applications must assure that the system is free from security vulnerabilities.

To obtain high assurance we recommend a hierarchical system architecture, where multiple layers provide specific, well-defined security mechanisms that can be used by higher layers. When a system is designed to provide a security mechanism, the mechanisms must be i) always invoked, ii) non-bypassable, iii) tamperproof and iv) evaluatable. Evaluation of the correctness of the mechanisms is a time consuming and difficult task, made even more difficult by complex security mechanisms.

Fortunately, a sound engineering approach (the MILS approach) exists to simplify the specification, design and analysis of security mechanisms. This approach is based on the concept of separation, as introduced by Rushby [232, 231]. The concept of separation has been accepted in the avionics community and is a requirement of ARINC 653 [52] compliant systems. Through separation, we can develop a hierarchy of security services, where each level uses the security services of a lower level to provide a new security functionality that then can be used by higher levels. Each level is responsible for its own security domain and nothing else. Limiting the scope and complexity of the security mechanisms provides us with manageable and more importantly, evaluatable implementations. General user applications can then execute, untrusted, isolated from each other except through communication channels managed by these security services.

In the remainder of this paper we provide an overview of the MILS architecture and its associated levels. In Section 8.2, we introduce the MILS architecture, the system design model that is the focus of our work. In Section 8.3, we present the partitioning kernel, the lowest layer of the MILS hierarchy. In Section 8.4, we discuss the hardware support needed to a separation kernel. Then in Section 8.5, we discuss the needs of shared device drivers. Section 8.6 expands on the basic MILS architecture for distributed systems. This is complemented by the discussion of MILS middleware and applications in Section 8.7 and Section 8.8. Sections 8.9, 8.10, and 8.12 conclude the paper with a description of an example MILS system.

8.2 MILS Architecture

In the past, secure systems were designed with the concept of a security kernel and a Trusted Computing Base (TCB) [95]. The key concept behind this approach is that the security decisions and the security enforcement mechanisms are an integral part of the TCB. Following this design paradigm, development teams found that more and more of their system functionality was being included in the TCB. Once this occurred, the evaluation of system security became unmanageable.

What is needed is a system architecture that allows a structured, compartmentalized approach to the design of a secure system. As we shall see, this design necessarily requires the deployment of several different execution environments (partitions/tasks) within a microprocessor. This design feature will force the system kernel to support many context switches per second – on the order of thousands. Fortunately, we are now at a point where the speed of current processors will support this level of context switches for the type of lightweight system kernel we propose. Evidence of progress in this direction can be seen in recent development efforts such as the ARINC 653 Standard [52] for partitioning operating systems, the Motorola AIM/Mask [179] separation-based operating system and encryption chipset, and the Integrity-178 [36] partitioning Real-Time Operating System.

The focus of this research is based on the concept of the MILS architecture, which was created to simplify the process of the specification, design and analysis of high-assurance computing systems [270].

Within the MILS architecture, application layer entities are provided with the mechanisms to control, manage and enforce their own application level security policies in a manner that ensures that the enforcement mechanisms are always invoked, non-bypassable, tamperproof and evaluatable. At this level we have trusted application-level security services and untrusted applications. The MILS architecture assumes certifiable trust within the microprocessor, separation kernel, middleware services layer, and for the application-level security services. Thus, we assume a benign fault model within the microprocessor and RTOS, but a malicious fault model for the untrusted applications. Benign faults will need to be addressed via traditional fault-tolerant diversity and redundancy, with fault containment procedures instituted within the microprocessor and RTOS. For example, illegal instructions and memory violations are assumed to be trapped and handled via the microprocessor and separation kernel. Likewise for covert channels and residual data needing sanitization. Violations of information flow that cannot be detected and trapped in this fashion will need to be handled within the middleware services layer. Faults occurring within the middleware services layer (those that cannot be detected and trapped at lower levels) constitute an open issue currently being addressed by the MILS research community. At the application layer, execution is confined to the application partition, with limited communication to other partitions. All communication is monitored by the certified application layer security services and lower processing levels. Applications are assumed to be rogue and are confined to operating within their partition resources, with finite boundaries of space and time. In the remainder of this section we introduce the concepts behind the MILS architecture.

8.2.1 MILS, MLS, MSLS and SLS

Traditionally, the military model of a secure operating system includes the concept of multi-level security (MLS). The idea behind this concept is that the system will be processing data items that are classified at different levels of security, and the information flow security policy that prevents the transfer of high-level classified information into low-level objects must be preserved. Therefore, we define a MLS system as one that must be certified to process and output co-mingled data at multiple classification levels. Classic security models, such as the Bell-LaPadula model [63], have been used to specify the secure behavior of such MLS systems.

The problem with full MLS systems is that they must be rigorously analyzed for security before they can be certified. Every portion of the MLS system must be analyzed to ensure that it properly handles labelled data and that there is no possible violation of the security policy. Even with a TCB architecture, or reference monitor [39], in place, there is often too much to evaluate.

The MILS architecture was developed to resolve the difficulty of certification of MLS systems, by separating out the security mechanisms and concerns into manageable components. These components are classified based on the way they process data:

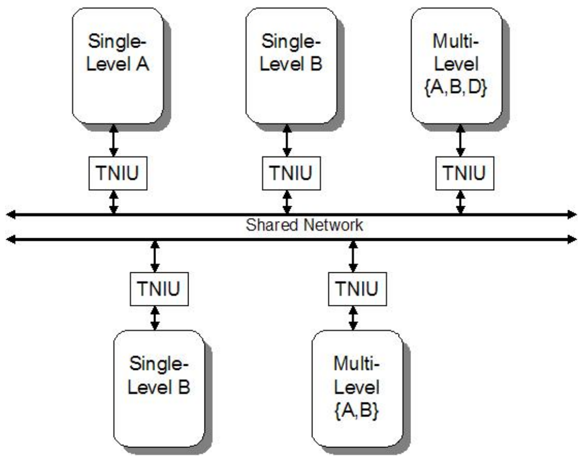

Figure 8.1: MILS refinement of an MLS system

- **SLS** Single-Level Secure component that only processes data at one security level.
- **MSLS** Multiple Single-Level Secure component that processes data at multiple security levels, but always maintains separations between classes of data. A device that processes messages one at a time (such as an I/O device driver) may be such a device.
- **MLS** Multi-level Secure components that co-mingle data at different security levels. Typically this is a device that will downgrade information from a higher level of security to a lower level through either filtering or the application of encryption technology.

A MILS system isolates processes into partitions, which define a collection of data objects, code and system resources. These individual partitions can be evaluated separately, if the MILS architecture is implemented correctly. This divide and conquer approach will exponentially reduce the proof effort for secure systems. To support these partitions the MILS architecture is divided into layers.

8.2.2 System Architecture View

Alves-Foss [26] defined a system architecture based on the separation of an MLS system into a MILS system consisting of multiple single level components with a few multi-level components. All components shared a common communication medium, (see Figure 8.1). In this work the author used the concept of logical trusted network interface units (TNIU) [230], to mediate communication between separate units. As mentioned in that paper, this mediation can be implemented by the operating system, in this case a MILS separation kernel (SK), which limits communication between partitions to only that which is specifically configured into the system. The system architect, using an SK, can graphically represent the system and authorized information flow.

The architecture in Figure 8.1 can represent a collection of SLS, MSLS and MLS components in a system, for example a collection of microprocessors or computers on a shared network. If the communication to the network is mediated by TNIUs, we get the configuration defined by Rushby and Randell [230], which was formally specified and verified to satisfy the restrictiveness [182, 181] security policy in [25, 31].

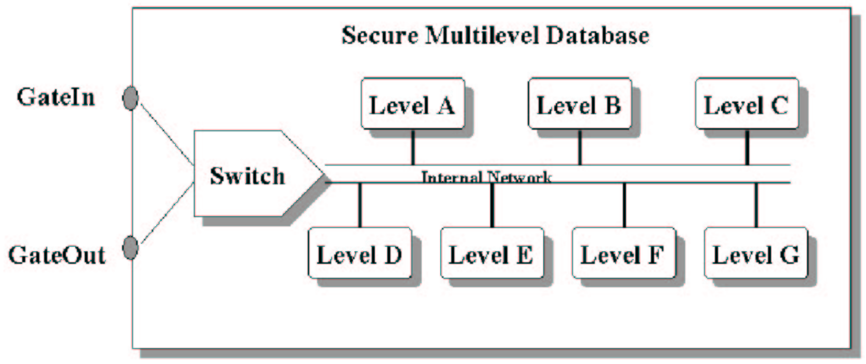

Figure 8.2: Whitebox Representation of Secure MultiLevel FileServer DataBase

However, we are not limited to a physical network or bus for communication. If we virtualize this network into representing kernel supported communication channels, then we get the type of separation system required by the MILS architecture, as specified in [26]. For example, we can take a multi-level file server and architect it as depicted in Figure 8.2. Regardless of the view, the architecture can refine MLS components down to a network of single level components, MSLS components, and simpler MLS components. With the SK and middleware in place, the architect can be confident of system security and safety. In addition, the certification of the components and layers is modular, allowing for great reuse.

At this level, the system architect views the system as a collection of execution engines, each of which processes data at one or more security levels, and limited communication channels that provide for information flow between the partitions. These channels may be shared memory segments, or SK-supported data streams. This view of the system can be depicted more abstractly as a directed graph with vertices for the partitions and edges for the channels; instead of the broadcast network show in Figures 8.1 and 8.2.

8.3 MILS Separation Kernel (SK)

The partitioner (separation kernel) layer is the base layer of the system, and is responsible for enforcing data separation and information flow controls within a single microprocessor; providing both time and space partitioning. This layer provides only a few base security mechanisms, following the recommendations of Saltzer and Schroeder [237] for economy of mechanism, keeping security mechanisms as simple as possible. The complexity of the partitioner is low enough that it can even be implemented in the microcode of a partitioning microprocessor as shown in [208]. The partitioner provides for the following:

1. *Data Separation.* The memory address spaces, or objects, of a partition must be completely independent of other partitions. The act of accessing an object by an executing partition must not affect the state of other partitions (no *exfiltration*), and an executing partition must not be affected by the state of any other partition or partition's objects (no *infiltration*).

2. *Information Flow.* This requirement is a modification of data separation. Although pure data separation would be easier to verify [231], it is not practical. There is a definite need

for partitions to communicate with each other. However, for secure systems, we need to be able to define the authorized communication channels between partitions. The SK will define precise moderated mechanisms for inter-partition communication. Only through these mechanisms may pure data separation be violated.

3. *Sanitization.* To ensure the information flow requirement, the SK is responsible for cleaning any shared resources (microprocessor registers, system buffers, etc.) before a process in a new partition can use them.

4. *Damage Limitation.* The consequences of a fault or security breach in one partition are limited by the data separation mechanisms. Addresses spaces of partitions are separate, and as such, an errant process in one partition can not affect processes in other partitions. The SK will also enforce bounds on shared resource, providing guaranteed minimum processing time, memory and other resources to the partitions as well as enforcing maximum usage of these resources.

In addition, the partitioner must be always invoked, non-bypassable, tamperproof and evaluatable. In the MILS architecture, the partitioner is responsible for timesharing the microprocessor between the partitions, and this function cannot be stopped. For high assurance systems satisfying the EAL7 certification requirements of the Common Criteria (CC), the functionality of the partitioner must be certified to have been rigourously verified using formal methods [53]. A CC protection profile has been developed to define the functionality of such a high-assurance partitioner [210], and submitted to the Open Group. One example of a partitioner specification can be found in ARINC 653 [52].

8.4 MILS Hardware Support

A system built on the concept of separation requires a certain amount of hardware support for the SK to work correctly. According to the Separation Kernel Protection Profile (SKPP) [210], and ARINC-653 [52] the hardware support for a SK includes:

- **Processing Power.** For any system, but especially real-time systems, the processor must have sufficient computing capacity to meet the worst-case timing requirements of the system.
- **Atomicity.** The processor must provide atomic operations for implementing processing control constructs, such as partition swaps and memory map changes.
- **Privileged mode of operation.** There will be some privileged instructions that must only be executed by the SK.
- **Memory Management Unit (MMU):** The MMU provides separation of address spaces between the partitions. Without hardware support for separation, there can be no data isolation or damage limitation. The processor must have access to the required memory resources and provide the SK with the ability to restrict partition access to memory.
- **Instruction Traps**. The processor must have some mechanism to transfer control to the SK if a partition attempts to execute a privileged or invalid operation.

- **Timing Control.** The processor must provide the SK with the ability to control and restrict the execution time of partitions; and therefore must have access to timing resources which provide a non-bypassable way of ensuring control is returned to the SK after some elapse of time.
- **I/O Access Limitation.** The processor must provide a mechanism for restricting access to I/O devices to specific partitions; and therefore have access to required I/O resources.

This basic list of processor features is available on many commercial microprocessors and motherboards; and should not be seen as a hinderance in the development of a secure MILS systems. The hardware used in MILS systems may be little different than the hardware currently used in embedded systems and can include commercial off-the-shelf hardware when available. For high assurance certification there may be requirements that the motherboards be certified, however this is not a specific requirement of the MILS architecture.

8.5 MILS Device Drivers

The MILS architecture requires that the SK remains small, to ensure that it can be fully evaluated. This means that services typically included in the operating system must now be included in the address space of the individual partitions, or delegated to separate "shared" partitions, with communications between partitions mediated by the SK policy. Device drivers for shared devices fall into this later category. Devices in a MILS system could include sensors, controllers and possibly mass storage.

When a device is private and should not be shared, it can be assigned to its own partition. Since most modern processors use memory-mapped I/O, the device can be protected from access by the MMU. An example of a critical device driver that should not be shared is the controller for the landing gear in an aircraft. Other less sensitive devices could be shared between several partitions. A sensor that places a sample in a device register is an example of a device that might be accessible to more than one partition. An engine temperature sensor would periodically update a data register with the latest reading and would need to be available to the process that updates the pilot's display as well as the engine overload warning system.

Devices in a high assurance system can also be considered critical and in need of privacy for security reasons. Intelligence agencies require that data at different classification levels not be stored on the same disk. Yet there is often a need to handle information from several classification levels. An MLS file system would need to separate data from partitions running at different classification levels and represents a shared storage device. The MLS file system would be isolated in its own partition with carefully defined communication paths between partitions at different classification levels. An example of a need for a shared storage device would be a military aircraft that must juggle data from radar, targeting activities and communication with traffic control.

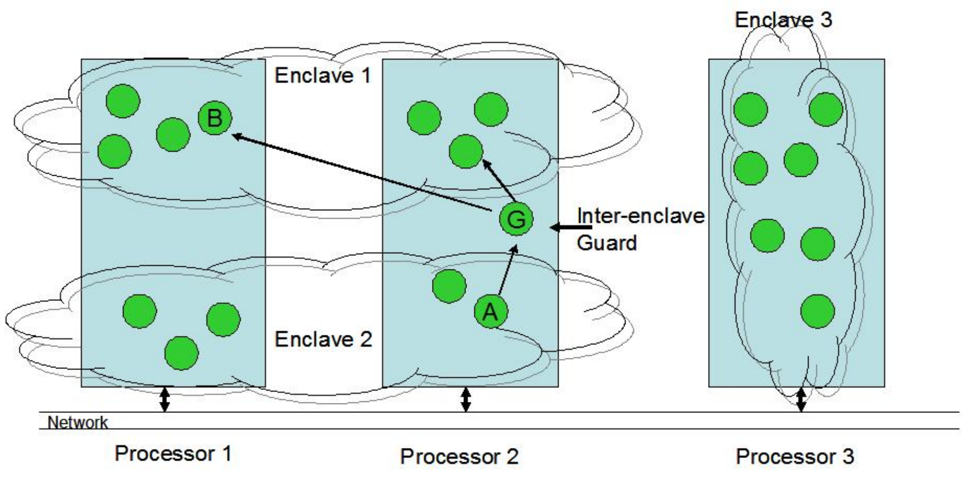

Figure 8.3: Enclaves distributed over separate processors.

8.6 Distributed MILS

To improve performance, individual partitions within enclaves can be mapped to separate processors (see Figure 8.3). An enclave is a group of partitions that are running at the same classification level. To provide MILS separation, this requires support at many levels. Of the four requirements of MILS systems, data separation, damage limitation and sanitization are no longer of concern between partitions on separate processors since there are no shared resources between these partitions. However, information flow controls must still be enforced.

Support for a distributed MILS system requires that the communication between processors be managed by the MILS system. If the inter-processor communication medium is open (i.e., accessible by non MILS managed components), then the communication must be cryptographically protected. However, if the medium is closed, then it may be possible to avoid the expense of encryption. If all processors run a MILS separation kernel, then the kernel can enforce network communication through an appropriate trusted device driver, as discussed in the previous section. If some processors in the system are running untrusted operating systems, we can use the TNIU approach, as discussed in Section 8.2.

Therefore, whether we have open or closed networks, all MILS components, or a mix of MILS and non-MILS components, we can architect the system to enforce the information flow requirements of the MILS system.

8.7 Middleware Services Layer

The middleware services layer provides for an extended scope of the separation concepts introduced by the partitioner. These services are dependent upon the needs of the specific application and are not constrained by the MILS architecture. They can include services such as resource allocation of shared data storage devices, object-oriented inter-partition communication, communication services between partitions on multiple processors, or real-time data distribution services. Middleware services are concerned about end-to-end data processing, and not just the single microprocessor

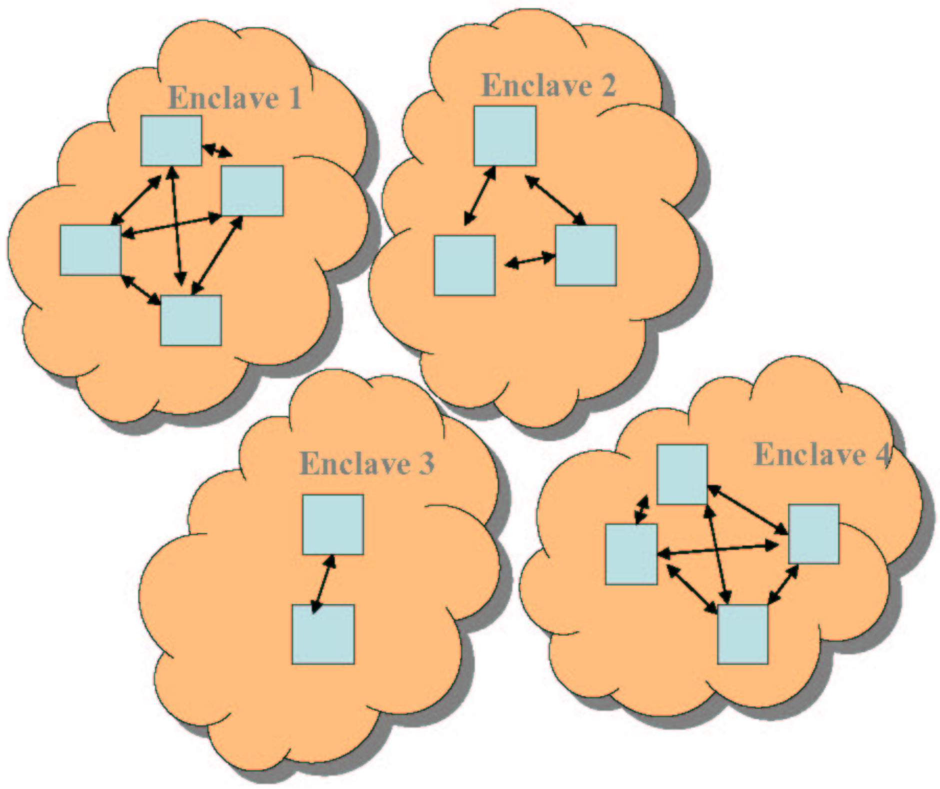

Figure 8.4: System of 4 independent secure enclaves

data processing of the partitioner. At the middleware layer, we begin to enforce the more traditional concepts of information flow. Each partition/address space in the system, no matter which microprocessor it is resident on, has a unique security label/classification. The system architect uses these labels to define the authorized communication between components. The labelling of the partitions and communication channels is used to satisfy the security policy. The middleware level is responsible for ensuring end-to-end security, through the following:

1. *Labelling.* The middleware layer must ensure that messages sent between individual partitions are correctly labelled with the sender's security classification and unique identity.

2. *Filtering.* The middleware layer is responsible for filtering out any messages that are not appropriately labelled before delivering them to the recipient.

3. *Maintaining Information Flow Controls.* The system architect designs the system with specific authorized information flow restrictions, and it is these restrictions that the middleware layer enforces.

At the middleware layer, we can introduce the concept of *authorized information flow.* If the system architect designs the system so that two partitions can communicate, then information flow between these partitions is authorized. This is true even if the two partitions are labelled with different security classifications. Middleware services provide secure information flows between partitions, typically via protocol-specific labelling and filtering. For example, a GIOP-guard would ensure that CORBA GIOP messages are formatted and routed correctly.

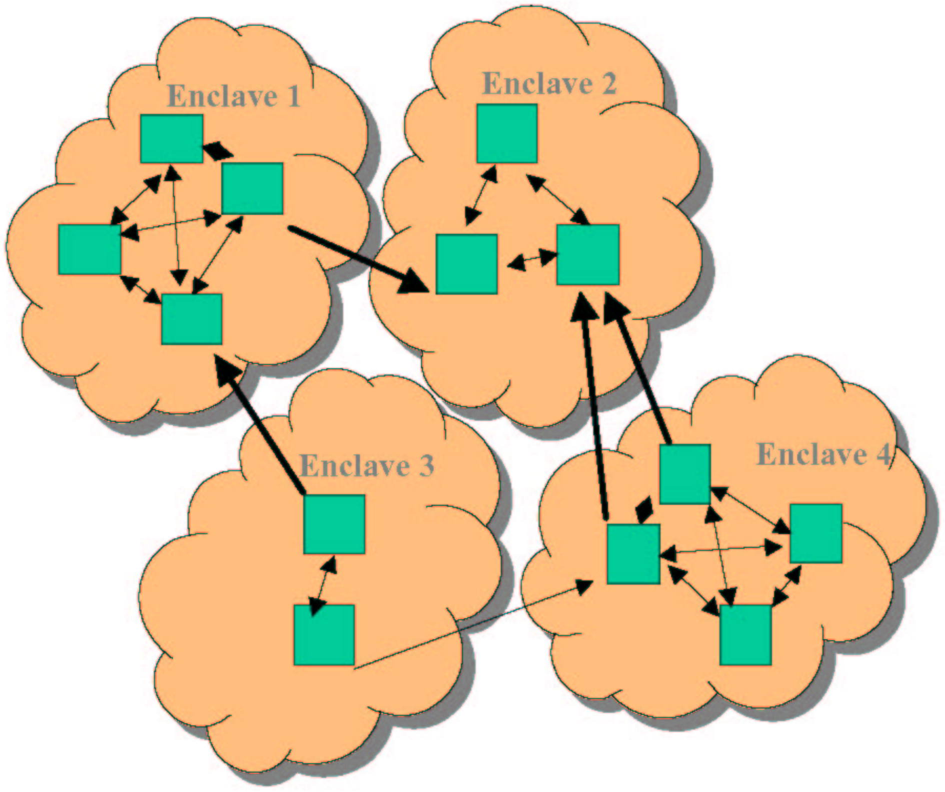

Figure 8.5: System of communicating independent secure enclaves

A system can be designed to be a collection of isolated enclaves, where partitions exist within a single enclave and there is no information flow between enclaves. If all partitions within an enclave are labelled with the same security classification, then we have a secure system as depicted in Figure 8.4.

Each partition of this configuration processes information at the security level of its enclave, and is called a *single level secure* (SLS) partition. This differs from a high-water mark system, which is a single level system in which all partitions process data at the same security classification, a configuration we wish to avoid, because they would all need to process the data at the highest level for it to be secure. If we wish to allow communication between enclaves, we can graphically specify that as in Figure 8.5, which will be discussed in more detail later.

8.8 Application Layer

The application layer is responsible for enforcing application layer security policies. Traditionally, MLS has been defined as data from more than one security classification. In MILS this definition has been refined into two distinct parts: multi-level secure (MLS) and Multi-Single Level Secure (MSLS).

An MLS component in a MILS system is one that deals with multiple classifications and transforms the data from one classification level to another. Because of the potential seriousness of violating its security policy, MLS components require the highest level of scrutiny and verification.

MSLS processes or devices also handle multiple data classifications but separate the data into independent streams with no communication between streams. Consequently, there is little danger that highly classified data will flow to unclassified entities assuming the MSLS device is functioning correctly. MSLS devices also need to be carefully verified that they maintain a separation secure environment, but they need a lower level of verification effort than the MLS components. Examples of these component types include the following:

1. *Collator.* A collator receives data from multiple classification levels, processes that data and transmits data at a single higher classification level. A collator is an example of a MLS device. This type of system is secure if the middleware layer can ensure that the transmission of information to the collator is one-way, that there is no feedback or response mechanism to signal the sender. If the middleware can be designed to truly support one-way communication, possible covert channels are automatically removed and this type of component is guaranteed to be secure.

2. *Downgrader.* A downgrader transmits data at a security classification level that does not dominate the highest level of input it receives. A downgrader, by definition, is necessarily an MLS component and must be independently evaluated for security. However, in the MILS architecture, the evaluation of the downgrader can be limited to processing within the downgrader's partition. We are already guaranteed data isolation and information flow control by the MILS architecture, and can take these properties as axioms for the higher layers. This greatly reduces system verification efforts. Further reduction results from careful system design as discussed in Section 8.2.2 when referring to the secure multi-level file system/database.

3. *Encrypter.* An encryption device is one where a plain text data stream is transformed into a non-readable encrypted stream according to a mathematical algorithm. Typically, data at a higher classification level is encrypted and sent out over a lower classified device. Thus, an encrypter is by definition an MLS device.

4. *MILS Message Router (MMR).* A MILS Message Router will function as a data switch by taking data from multiple partitions at various classification levels and routing the messages to the correct destination, which may include additional trusted devices that determine if the message satisfies the application-level security policy. Messages will be checked to insure authorization exists for communication between the partitions. Since transformation between classification levels is not being performed, the MMR is an example of a MSLS device.

The ability to isolate the portions of the system that necessarily process MLS data enables us to focus our resources and limit our verification efforts, making high assurance components realizable. The system architect effectively builds into the system a set of *software firewalls* that are responsible for implementing the application layer security policy with confidence that the middleware and SK will ensure that the application layer mechanisms are non-bypassable, tamperproof, always invoked, and evaluatable.

The MILS architecture now permits us to reuse the layers. We can port a middleware package from one certified SK to another with greatly reduced additional proof effort. We can avoid the process where each verification effort restarts from scratch; instead we reuse the certified layers in the architecture.

8.9 A Secure System Using MILS

Consider again the system depicted in Figure 8.3. If partition A in *Enclave 2* wishes to send a message to partition B in *Enclave 1*, we need to be sure that the message sent does not violate the security policy. There are two scenarios we need to investigate:

1. $A \leq B$. In this configuration, the security level of A is less than that of B. In other words, information is permitted to flow freely from A to B. Since the MILS system enforces information flow policies dictated by the system configuration, this flow is controlled by the SK and the Distributed MILS system.
2. $A \nleq B$. In this configuration, the security level of A is not less that the security level of B. Now, information is not permitted to flow freely from A to B. Given this restriction we have two choices:
   - The first is that A has been evaluated and certified to be *trustworthy*. This means that it has been shown that any information A does send to B is not in violation of the security policy. For example, A could be a cryptographic engine which is certified to encrypt any messages it sends to B.
   - The second is when A is not trustworthy. In this case we need a trusted intermediary (a guard) to control information flow from A to B. This guard is a trustworthy application which is responsible for analyzing the content of the communication and determining whether this communication is in accordance with the system security policy. The guard has the ability to modify the contents of the message, delete the message or send a constructed response back through the MMR.

8.10 Example MILS System

We are not limited to having each node of the graphs of Figures 8.4 and 8.5 represent a partition. For example, in a shared-memory system, the system can be viewed as the directed graph depicted in Figure 8.6, where partitions are only connected to memory segments and memory segments are only connected to partitions. Interprocess communication is implemented through reading and writing of shared memory buffers[1]. Partitions can read or write a connected memory segment depending on the direction of the edges.

In this example, the system depicted is a secure crypto communication device. This device receives data from the *red* network on the left, through the network interface (RPM). This data is assumed to be correctly labelled. Based on the labelling the system sends the data to one of three encryption engines (Type 1 devices), each implementing a single encryption algorithm,

[1]In general, MILS inter-partition communication may occur through any verified communication channel offered through the kernel.

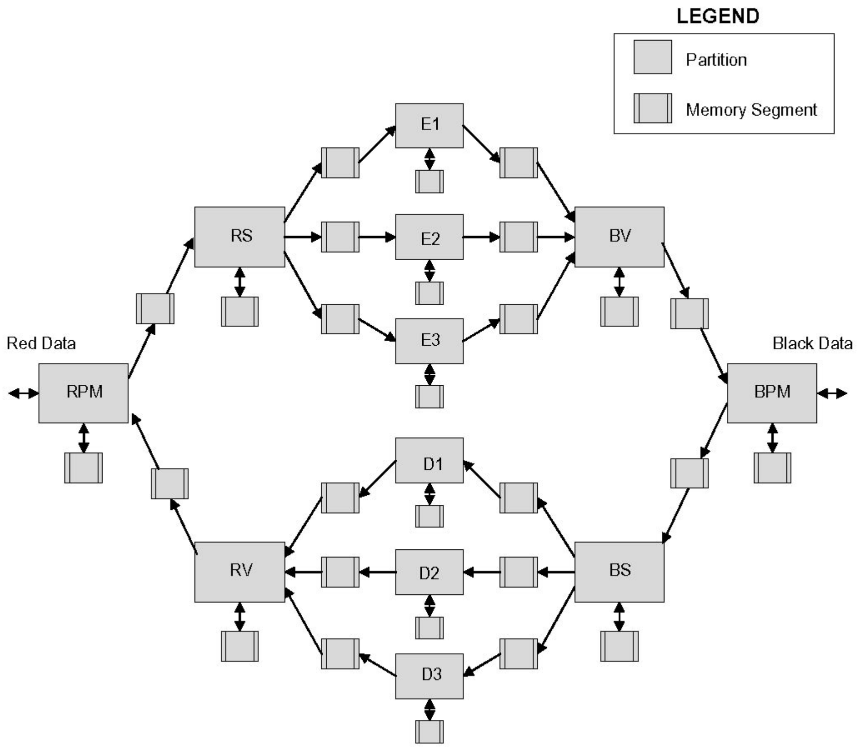

Figure 8.6: System Graph for Shared Memory Architecture

and maintaining separate encryption keys. The encrypted messages are then transmitted on the insecure *black network*. Incoming encrypted messages from the black network are appropriately sent to the correct decryption engine. The results are then labelled correctly and sent out over the red network.

The MILS system depicted in this figure forces isolation between partitions. The only authorized information flow occurs through the shared data segments as depicted in the graph. Input into the Red Switch (RS) is separated multi-level data, therefore the RS device is an MSLS application that enforces the system security policy by correctly transmitting data to the appropriate shared memory buffer for the encryption engines. Similarly the Red Verifier (RV) receives single level information from the decryption engines and combines it to transmit out on the red network (appropriately labelled). This device is also necessarily an MSLS device. Given the existence of a verified MILS SK, the only two components which need verification are RS and RV (possibly RPM depending on how security labels are managed). Every other component is single level, or is a separately verified Type-1 encryption device, and does not need security verification. This reduction in verification effort was capitalized on by the AIM system developed by Motorola, which uses the MASK separation kernel [179].

8.11 MILS Research

8.12 Conclusion

High assurance systems, whether they be security critical or safety critical, require an extensive amount of analysis. In this paper we have discussed the MILS architecture, a design approach for high assurance systems that enables manageable analysis through modular design and layered enforcement of security policies. At the lowest level of enforcement, the MILS architecture supports the policy of data isolation, information flow, damage limitation and sanitization. Specific higher level security policies must be enforced by trusted partitions utilizing the resources of lower level policies. The lower levels support the enforcements mechanisms of the higher levels, working as partners to support the application level security policy.

The MILS architecture is not just an academic exercise, but rather an approach to system design that is supported by industry and government. Partitioning kernels, the lowest layer of the MILS architecture are already being deployed by multiple real-time operating system vendors [36]. Common criteria protection profiles are currently being developed for both the partitioning kernel [210] and MILS middleware. At the University of Idaho, we are currently developing a testbed for MILS concepts, which had been built, in its entirety, from COTS components. The testbed consists of four single board computers and the Integrity and LynxOS-178 RTOS operating systems.

In addition, there is much work currently being done on the necessary formal methods for efficient mathematical modelling and analysis of high assurance systems. We have already developed proofs for the separation maintained by an exemplary separation-based microprocessor [208]. Greve et al. developed a security policy for separation kernels and have shown how it can be used in a two-level hierarchy [121]. We have evaluated this work and discussed the uses and limitations of it [34]. Results from our testbed and formal proofs will be forthcoming in publications from the authors.

Chapter 9

Hidden Implementation Dependencies in High Assurance and Critical Computing Systems

9.1 Introduction

When developing High Assurance and Critical Computing Systems (HA&CCS), compelling evidence is required that the system under development satisfies certain critical properties while achieving its functional objectives. Both functional and nonfunctional requirements contribute to system correctness; a system is correct if it correctly implements every functional requirement while complying with every nonfunctional requirement.

HA&CCS must involve rigorous development, evaluation, certification, and auditing processes. The evidence for such rigorous evaluation and certification processes must be collected and maintained during the analysis, design, development, and maintenance processes. In the end, these processes must result in a system implementation along with a set of arguments to support the claim of system correctness based on collected evidence, with a high level of assurance.

Relationships between artifacts generated by these processes form an essential component of this evidence, we refer to them as *traceability relationships*. These enable incremental verification and validation of correctness and dependability properties. In addition, they enable analysis of compliance with required standards (e.g., IEC 61508 [2] and DO-178B [228]) and subsequent certification audits during the lifespan of a system.

Omissions in requirements, poor understanding of interfaces, and ineffective communication between *stakeholders*[1] are key causes of safety-related errors in HA&CCS [158, 156, 167, 157, 125, 155, 268, 166, 87]. Such errors may result in critical failures with very costly or catastrophic consequences. Misunderstandings, omissions, and ineffective communication between stakeholders

[1]The term *stakeholder*, in the context of this article, should be understood as a stakeholder of the system model (set of work product sections), which means: any person involved in the engineering, design, construction, certification, operation, maintenance, modification, and auditing of a system. Therefore, stakeholders have a need to know information about a system, its operational environment, and the model used for its development, e.g., requirements, environmental constraints, architectural descriptions, and configuration settings.

can be attributed to sociological, organizational, cognitive, and/or technical reasons [158, 156, 155].

One way to help avoid some of these misunderstandings and omissions is to give stakeholders the ability to navigate and visualize a system model along with its associated rationale and other related information. In other words, we need to have efficient and complete *traceability of work product sections*. We define a **work product section (wps)**[2] as a semantically meaningful section or unit of a work product, where we use work product as defined by the Software Engineering Institute (SEI®) [244]. In addition, we define **traceability of work product sections** as the ability of a stakeholder to manually or mechanically describe and navigate relationships between work product sections.

Such an approach to offering total information availability necessarily implies the following assumptions: I) Work products are existent, and their sections are uniquely identifiable, addressable, and stated in a machine-readable format. Also, there are tools for the creation and maintenance of work products and their sections. II) Traceability relationships between wps(s) can be created and maintained in a machine-readable format between desired wps(s), and there are tools for their creation and maintenance. III) There are tools for the visualization, navigation, and retrieval of work products sections, as well as their associated traceability links. We explain the rationale behind these assumptions in Section 9.3 of this article.

The problem we still face, even under these three assumptions, is that of enabling stakeholders to visualize hidden relationships or dependencies in order to aid them with the discovery of hidden safety hazards. Even when developing HA&CCS following the most rigorous standards, where traceability of wps(s) is assured by the creation and maintenance of traceability links, there are still unforeseen dependencies that arise from the complexity of these systems and the high cohesion of software [158]. Such hidden dependencies, which are very difficult to uncover, may result in unfounded assumptions of independence between system wps(s), which in turn may turn into unsafe interactions, which may lead to critical or catastrophic failures.

As demonstrated by the two case studies presented in this article, these assumptions, which may result from unknown or unconsidered relationships (dependencies) between wps(s), can lead to critical or catastrophic failures. Effective and complete traceability may help reduce the causes of critical or catastrophic failures by exposing those hidden dependencies.

In this article, we briefly introduce a framework where Assumptions I, II, and III are valid (Section 9.3), followed by a technique for the mechanical discovery of hidden implementation dependencies between wps(s). The technique consists of three steps: a) building a formal implementation model of wps(s) and their associated abstraction levels for a given system under analysis (Section 9.4), b) enforcing strict partial order properties in the formal implementation model of wps(s) (Section 9.5), and c) applying the conceptual completeness principle to the formal implementation model of wps(s) (Section 9.6). We demonstrate how this technique helps uncover hidden implementation dependencies by applying it to two case studies, which are described in Sections 9.7 and 9.8.

The two case studies are: 1) the Minimum Safe Altitude Warning (MSAW) system and a related aircraft accident at the Guam International Airport in 1997 in which 228 people died (Section 9.7) and 2) the Guidance and Control Software (GCS) project data, based on a hypothetical landing

[2]In this article, we use the term *wps* to denote one work product section and the term *wps(s)* to denote more than one.

spacecraft module developed by the National Aeronautics and Space Administration (NASA) for use by the Federal Aviation Administration (FAA) (Section 9.8).

We do not believe that an implementation of the framework and the application of the technique described in this article are sufficient conditions for avoiding all accidents due to misunderstandings and miscommunication between system model stakeholders. Misunderstandings and communication problems could still arise from ineffective organizational communication channels and the hazards associated with enterprise control structures. We do argue that effective traceability and navigational aids are an essential aspect of a systems-oriented approach to engineering high assurance and critical computing systems, that they facilitate the flow of information between stakeholders, and therefore, that they would help minimize errors that may develop into critical or catastrophic failures.

Section 9.9 reviews related work. An analysis of the scope of this research along with a restatement of our contribution is presented in Section 9.10. Before fully developing our methodology, we briefly introduce HA&CCS and the main causes of critical errors in them (Section 9.2).

9.2 Errors in High Assurance and Critical Computing Systems

In this section, we introduce HA&CCS and report on studies which show that most safety-related failures in these kind of systems are not due to errors occurring in single components, but on the contrary, they are due to complex and unexpected interactions between system components (wps(s) of the system model). We also present a brief introduction to the concept of *emergent properties* and how it relates to the work contributed by this article.

9.2.1 Introduction to HA&CCS

High Assurance Computing Systems (HACS) are systems for which compelling evidence is required that the system delivers its intended services while satisfying certain required critical properties [104, 185]. In other words, HACS are systems where rigorous or formal verification (compelling evidence) of functionality (delivery of intended service) and of emergent properties (satisfiability of critical properties) are required. Among the required properties which HACS must satisfy are security, safety, real-time, and dependability properties [104, 185]; these are also required by critical systems.

Critical systems are systems where great losses or damage, either to life, the environment, economic stability, the mission, or the system itself, could result from unexpected behavior (adapted from [134]). Incorrect operation could be, for example, failure to abide within safety restrictions or failure to deliver minimum critical services when required [56, 57, 149]. There are four approaches to developing critical systems which stem from different engineering disciplines: dependability, safety, security, and real-time [233].

We use the term **high assurance and critical computing systems (HA&CCS)** to refer to both classes of computing systems. The differences in the literature between labeling of computing systems as *high assurance* or *critical* is that *high assurance* explicitly requires compelling evidence of correct functioning while *critical* states that malfunctions could cause major losses. We believe that high assurance computing systems are, in fact, critical systems, and critical systems require, given their nature, high assurance of correctness. Both classes of systems must be developed and maintained using high levels of assurance and quality in organizational, process, and project standards. They also must be developed, maintained, and operated through utilization of appropriate state-of-the-art tools and scientific methodologies to their maximum extent in order to ensure correct operation of the system and absence of damage to humans and the environment. In particular, the technique described in this paper is concerned with preventing safety-related errors that can occur in HA&CCS.

9.2.2 A Key Cause of Safety-Related Errors in HA&CCS

In a recent article, Lutz and Mikulski state, as two of five recommendations for improving safety in spacecraft systems, that it would be necessary to: 1) maintain traceability of operational procedures back to their originating requirements and 2) integrate requirements engineering within the system's maintenance and operational processes [167].

In the early 1990's, Lutz analyzed, in detail, 209 safety-related errors (having a significant or catastrophic effect on the mission) in safety-critical embedded systems [166, 165]. These 209 safety-related errors comprised 54% of the 387 errors discovered during system testing and integration for the software of the Voyager and Galileo spacecrafts. As a result of this analysis, Lutz reports that "difficulties with requirements" (e.g., discrepancies, omissions, and misunderstandings) are a key cause of errors that threatened the missions.

In recent years, Leveson and colleagues at the Massachusetts Institute of Technology (MIT) conducted an in-depth analysis of reports of catastrophic accidents that occurred with three aircrafts and five spacecrafts. Leveson describes three categories of common factors existing among accidents of both classes (spacecraft and aircraft): 1) flaws in the safety culture, 2) ineffective organizational structure and communication, and 3) ineffective or inadequate technical activities. Within each category, she describes and explains several systemic factors that contributed to the accidents, three of which are: limited communication channels and poor information flow, inadequate specifications, and conflicting and inadequate documentation [158, 156, 159, 155]. Leveson states in her latest book draft entitled *System Safety Engineering: Back To The Future* [158], page 271 as of June 2006:

> Almost all the software accidents known to the author have resulted from the computer doing something wrong rather than the computer hardware or software failing to operate at all. In fact, each of the software or hardware components may have operated according to its specification (i.e., they did not fail), but the combined behavior of the components led to disastrous system behavior.

The common characteristic in all of these accidents is that they are the result of unexpected interactions between system components. We believe that an essential step in the path to anticipating these unexpected interactions is to make every possible interaction (dependency) between system components, requirements, and safety constraints (wps(s)) accessible to stakeholders. The main objective of this work is to discover and make visible these hidden dependencies between wps(s), based on a defined initial set of traceability links between wps(s) and their respective levels of abstraction, and therefore enable stakeholders to observe relationships between components (wps(s)) that could lead to unsafe emergent behavior.

9.2.3 Emergent Properties in Complex Systems

The notion of emergence has been investigated by philosophers since at least the 18th century [89]. Since then, there have been several approaches to the analysis and definition of the concept. The Stanford Encyclopedia of Philosophy presents a good introduction to the history and definitions of the concept: "Emergent properties are systemic features of complex systems that could not be predicted (...) from the standpoint of a pre-emergent stage,..."[278]. The notion of emergence is intrinsically related to the notion of levels of abstraction in the sense that an emergent property "emerges" into a new level of abstraction but it cannot be observed at any of the previous levels. Popper and Eccles, when writing about "Indeterminism; the Interaction of Levels of Emergence", state, "Each level is open to causal influences coming from lower *and* from higher levels" [218], and Damper states, "...emergence is best considered from the perspective of the *understanding* which can stem from viewing complex phenomena and systems at different levels of abstraction, as opposed to the *difficulty* or *impossibility* of so doing" [89]. The safety of a system, as well as most dependability properties, are necessarily emergent properties of a system.

9.3 A Framework for Effective Traceability

Due to the complexity of today's high assurance and critical computing systems, it is currently very difficult for any stakeholder to anticipate and visualize unsafe interactions between system components (wps(s)). In order to analyze dependability properties in a system model, due to their emergent (or systemic) characteristic, it is necessary for stakeholders to position themselves at a different level of abstraction from the one used to model those wps(s) (components) being analyzed. Moreover, mechanical aid to help cope with this problem is limited due to the current lack of semantically rich work product representations (also called thick descriptions by Ramesh and Jarke [219]).

For example, informal requirements are treated as a block of text describing a desired behavior without a machine-readable statement about the objects or subjects involved in such requirements and description of physical units, valid ranges, and accuracy associated with all physical measurements are literally hundreds of pages away from where dependent constants or variable values are defined and used (e.g., in configuration files or in source code).

We believe that, in order to leverage the use of mechanical aids that facilitate the task of humans in the discovery and analysis of dependencies that could lead to unexpected emergent behaviors, we need to give stakeholders complete and effective traceability, visualization, and navigational aids. These functionalities can only be achieved by the implementation of a framework and tools where multidisciplinary informal and formal methodologies and representations can coexist and interact.

Such a framework is composed of: I) semantically rich and machine-readable representations for all wps(s) written in any system specification language used in a project (including for example, requirements, test cases, and configuration files), where unique identification of work products is available, II) semantically rich and machine-readable languages for the representation of traceability links between desired wps(s), along with tools for the creation and management of those links, and III) availability of tools for the creation, management, visualization, and navigation of represented wps(s) by using their associated traceability links.

The implementation of such a framework, including supportive tools for the creation, maintenance, management, and visualization of system and software engineering data, is an enormous endeavor that cannot be accomplished by an individual nor a small team, but it requires the cooperative work of the research community and industry in the domain.

Fortunately, the research community and industry are already cooperating toward this goal and have been working, with different approaches, toward achieving some of the objectives needed. With the purpose of showing the reader that this is a trend of work and that the goal stated in this section will very likely be achieved in the near future, we cite some of that work.

One solution to the implementation of this framework that has been gaining momentum in the research community, and industry as well, is the use of the Semantic Web model [66] and its associated open technologies (e.g., OWL, RDF, SVG, URI, XLink, XML, XSL) for the representation of wps(s) and their associated traceability relations. A wealth of information about the Semantic Web and its associated technologies, including the corresponding recommendations, can be found in the World Wide Web Consortium web site [18]. There are also several authors, research groups, and industry organizations which have developed XML-based representations for wps(s) and their associated transformation tools. Examples are: JavaML [58, 175], CppML [175, 9], OOML [175], ZML [251], MathML [1], ELotosML [85], XMI [5], AADL [236], rich requirement representations and navigation [238, 105], and RML [85]. In addition, XML-based representations for traceability links have been developed such as TraceML [85, 29] and Xlinkit [205, 202, 203].

With respect to the visualization and navigational aids, there are also a number of research groups that have developed knowledge visualization tools, both domain specific and generic knowledge visualization. Take for example the CHISEL group and their *SHriMP*, *Creole*, and *Jambalaya* tools [7], the Bauhaus group and its *Bauhaus Toolkit* [13], and the Software Engineering Research Laboratory with the *Chimera*, *InfiniTe*, and *KAS* tools [16].

We believe that it will not be long before we have access to a complete implementation of this framework, either by using the Semantic Web implementation approach, or a different approach with similar purposes and objectives, where Assumptions I, II, and III presented in the introduction of this article hold true and where representation, search, linking, and retrieval facilities for wps(s), as well as visualization and navigation facilities (for wps(s) and traceability links) are part of the state of the practice in systems and software engineering organizations.

Hence, for the rest of this article, we assume that such a framework, where Assumptions I, II, and III hold, has been implemented and is in use. Under these assumptions what is left to do, and the main contribution of this article, is to develop methodologies for aiding stakeholders in the discovery of potential causes of failure due to unforeseen, and potentially unsafe, emergent behavior.

9.3.1 An Expert System for System Models (SyModEx)

For the purposes of our research, we have developed a proof-of-concept expert system named **SyModEx**. SyModEx uses Visual Prolog [17] to implement the formalization described in Section 9.4 of this article, and it is capable of mechanically deriving hidden dependencies using the technique described in this article. SyModEx is also capable of transforming the underlying semantic network into a graph representation using the *dot* graph language [111] and the Graph eXchange Language (GXL) [271, 10]. *dot* graph files can be read and rendered by the Graphviz graph layout application [112, 11], among other tools. GXL graph files can be drawn and edited by the SHriMP artifact navigation tool [193, 7], among other tools. The graphs shown in Figures 9.2 and 9.4–9.6 were created by SyModEx as *dot* graphs, and the graph layouts have been generated by Graphviz. The graphs from Figures 9.7–9.9 were created by SyModEx as GXL graphs and drawn using the SHriMP tool [193]. In a Semantic Web-based implementation of our framework, the semantic rules, currently expressed as Prolog predicates, would be represented using a language such as RuleML [15] and processed by a reasoner such as *KAON2* [198, 12].

9.4 Formal Traceability

In this section we present a formal theory Γ_{Theory} for the representation of wps(s), their associated traceability relations, and the structural properties expected for each traceability relation. This formal theory is based on a restricted version of a *dyadic first order predicate calculus* [199] and can also be defined as a *mathematical system* [243].

We also define a formal model Γ_{Model} which gives a semantic interpretation of the formal syntax defined by Γ_{Theory}. Based on this model, we loosen the distinction between the syntactic and the semantic portions of our system by adopting some renaming conventions. We use the symbol Γ to represent both the theory and the model.

The mathematical system Γ_{Theory} is composed of 1) a formal syntax, 2) a set of axioms, and 3) a set of inference rules. The formal model Γ_{Model} defines a formal semantics for Γ_{Theory}.

9.4.1 Formal Syntax of the Theory

The formal syntax of Γ_{Theory} is defined by its formal language Σ, which is composed of all well-formed-formulæ (*wff*) that can be generated by using the alphabet Λ (Eqn. (9.1)) and the formation rules presented in Eqn. (9.2).

$$\begin{aligned}
&\Lambda = S \cup C \cup K \cup E \cup P. \qquad (9.1)\\
&S = \{s_1, s_2, \cdots, s_n\}.\\
&C = \{l_1, l_2, \cdots, l_q\}.\\
&K = \{\neg\}.\\
&E = \{(,), ,\}.\\
&P = \left\{P_1^2, P_2^2, \cdots, P_p^2\right\}.
\end{aligned}$$

$$\omega = P_t^2(\sigma_i, \sigma_j) \in \Sigma \Leftrightarrow (\sigma_i, \sigma_j \in S \cup C) \land P_t^2 \in P. \tag{9.2a}$$

$$\omega = \neg P_t^2(\sigma_i, \sigma_j) \in \Sigma \Leftrightarrow (\sigma_i, \sigma_j \in S \cup C) \land P_t^2 \in P. \tag{9.2b}$$

$$\omega \in \Sigma \Leftrightarrow \omega \text{ is derived only from the application of rules (9.2a) and (9.2b).} \tag{9.2c}$$

The alphabet Λ is the set of symbols defined in S, C, P, K, and E, where: S is a finite set of subject symbols, C is a finite set of characteristic (or property) symbols, P is a finite set of dyadic (second grade) predicate symbols, E is a set of delimiter symbols, namely: right parenthesis, left parenthesis, and comma, and K is the singleton consisting of the unary negation operator.

The two formation rules given by Eqn. (9.2a) and Eqn. (9.2b), along with the closure rule (Eqn. (9.2c)) allow us to build binary predicates, and their negation, as well-formed-formulæ of Σ. Notice that in this definition, the set of symbols Λ is finite and there are no recursive formation rules. Therefore, all well-formed-formulæ of the language Σ are atoms and Σ is finite, with $|\Sigma| = 2 \cdot p \cdot ((n+q)!/(n+q-2)!)$.

9.4.2 Types of Traceability Links and Predicates of the Theory

There are different types of relationships between wps(s), and there is no *de-facto* or published standard for the nomenclature of those relationships.

Pohl describes a taxonomy of traceability link types and lists five first-level and 18 second-level types of links that were found after analyzing approximately 30 different publications on the topic of requirements traceability [217]. Pohl's five first-level traceability link types are: *Condition, Content, Documents, Evolutionary, and Abstraction*. In this last category Pohl lists two second-level traceability link types: *Refines* and *Generalizes*, which closely match what we call the *partial implementation relation*.

Ramesh and Jarke describe and classify types of requirement traceability links in four high-level categories which they call *Satisfaction*, *Evolution*, *Rationale*, and *Dependency* [219]. The category called *Satisfaction* corresponds to our partial implementation relation; or for short, **implements**, which is analyzed in Section 9.5.

9.4.3 Formal Semantics of the Theory

Based on Assumption I, we assume that for a given project there exists a set of wps(s) which are uniquely identifiable and addressable; we will call this set $\mathbb{S}$. We also assume that there is a set of characteristics that can be associated with any wps. For example, we could say that a wps is *critical* or that it belongs to a given level of abstraction; we call this set $\mathbb{C}$.

Based on Assumption II, we will assume that types of traceability links are determined and unique; we construct a set of traceability relationship types $\mathbb{T}$. Predicates in our formal model Γ_{Model} will correspond to types of traceability relations.

The semantics of a predicate calculus are given by 1) an *Interpretation* plus 2) an *Assignment*. An Interpretation consists of the following: a semantic domain, functions from each set of symbols representing individuals to the semantic domain, and a meaning for each predicate symbol. An *Assignment* is a function from the set of variables to the semantic domain [199].

The formal semantics of Γ_{Model} are given by the interpretation I as defined by Eqn. (9.3). An *Assignment* function is not needed in Γ_{Model} because the language Σ has no variable symbols.

$$\begin{aligned} I &= \langle \mathbb{D}, f_S, f_C, f_P \rangle \,. \\ Where : \mathbb{D} &= \mathbb{S} \cup \mathbb{C} \; and \\ & f_S : S \rightarrow \mathbb{S}, f_C : C \rightarrow \mathbb{C}, f_P : P \rightarrow \mathbb{T}. \end{aligned} \tag{9.3}$$

Semantic Domain: $\mathbb{D}$ is the semantic domain (also called base domain) where $\mathbb{S}$ is the set of all uniquely identifiable and addressable work product sections for a given system model and $\mathbb{C}$ is the set of all characteristics of wps(s).

Work Product Sections Semantic Function: We define the function $f_S : S \rightarrow \mathbb{S}$ as: $f_S(s_i) = x_i$, where s_i is the i-th symbol of S, and x_i is the i-th wps of $\mathbb{S}$.

Levels of Abstraction Semantic Function: For the purposes of this article the set C of symbols will model only the levels of abstraction selected for a given project. Hence we will define $\mathbb{C} \equiv \mathbb{A}$, where $\mathbb{A}$ is the set of abstraction levels for a given project. We define the function $f_C : C \rightarrow \mathbb{C}$ as: $f_C(l_i) = \alpha_i$, where l_i is the i-th symbol of C, and α_i is the i-th level of abstraction defined in $\mathbb{A}$. Other characteristics (properties) of wps(s) can be formalized using the same formalism.

Predicate Symbols Semantic Function: For the purposes of this article we define the set of traceability relationship types as: $\mathbb{T} = \{$ *implements*, *impDependency*, *abstractionLevel*, *higherAbstractionLevel* $\}$. We define the semantic function $f_P : P \rightarrow \mathbb{T}$ as $f_P(P_i^2) = t_i$, where t_i is the i-th traceability relationship type of $\mathbb{T}$.

9.4.4 Renaming Conventions

Without loss of generality, and in order to make the nomenclature more representative of the semantics assigned to each well-formed-formula, we adopt the following renaming conventions for our formal model Γ.

Work Product Sections Renaming: We will rename each symbol s_i in S with the identifier of the wps x_i assigned to s_i by f_S.

Abstraction Levels Renaming: We will rename each symbol l_i in C with the identifier of its corresponding abstraction level assigned by f_C.

Predicates Renaming: We will rename each predicate symbol P_i^2 with the unique identifier of the traceability relationship t_i in $\mathbb{T}$ assigned to P_i^2 by the semantic function f_P.

Following our renaming conventions, the well-formed-formula *implements* (*IncreaseVelocity-*

Code, VehicleAccelConstraint) indicates that the wps named *IncreaseVelocityCode* partially implements the wps named *VehicleAccelConstraint*.

9.4.5 Formal Traceability Links and Axioms of the Theory

Using our formal model, we formalize a traceability link with a binary predicate $P_t^2\,(s_i, s_j)$ between two wps(s), indicating that there is a relationship of type t from the wps s_i to the wps s_j (Eqn. (9.4)).

Definition 23. *A **traceability link** is an axiom or theorem of* Γ_{Theory}.

$$P_t^2\,(s_i, s_j) \text{ is a traceability link} \Leftrightarrow \Gamma_{Theory} \vDash P_t^2. \tag{9.4}$$

The axioms and theorems of Γ_{Theory} are those predicates that represent a traceability link in our system model. In other words, a well-formed-formula $\omega = P_t^2\,(s_i, s_j)$ is *valid* under the interpretation I if ω exists as a traceability link.

Given a formal model Γ, the set of all axioms and theorems of the form $P_t^2\,(s_i, s_j)$ of a given type t defines a **traceability relation**. Each axiom/theorem defines an ordered pair of a binary relation $t : \mathbb{S} \cup \mathbb{C} \rightarrow \mathbb{S} \cup \mathbb{C}$. For example, the set of all *implements* predicates defines the *partial implementation* traceability relation.

9.4.6 Rules of Inference of the Theory

In a formal system, the rules of inference allow us to derive new theorems based on current axioms and theorems. In our formal system Γ, axioms represent *given* or known traceability links and theorems represent *derived* traceability links.

We define for Γ three groups of inference rules. The inference rules in these groups were developed with the purposes of formalizing the partial implementation traceability relation and of discovering hidden dependencies between wps(s). The first group corresponds to the strict partial order for the partial implementation relation. The second group corresponds to the strict partial order for abstraction levels and the relationship between wps(s) and abstraction levels. The third group corresponds to the conceptual completeness principle. These groups will be described in detail in Sections 9.5 (Groups 1 and 2) and 9.6 (Group 3).

When defining inference, rules we use the symbols x_i, y_j, z_k, α_p, α_q, α_r as variables to represent individuals of the semantic domain $\mathbb{D}$; we will also use standard logic symbols such as $\forall, \exists, \in, \rightarrow, \wedge, \vee$. These are symbols of the meta-language and are not to be confused with the symbols in the alphabet Λ of Γ.

9.4.7 Partial Order Relations

Before beginning the next section, we briefly introduce partial orders. A binary relation from a set A into a set B is a subset R of the Cartesian product $A \times B$. A binary relation $R_{S\times S}$ is a *strict* or *irreflexive partial order* in S if and only if R is *irreflexive* and *transitive.* R is called a *weak* or *reflexive partial order* if it is *reflexive* and *transitive.* The former represents a relation of the $<-$class and the latter represents a relation of the $\leq -$class [161]. Since we are assuming S is a set, the structure $\langle S, R_{S\times S}\rangle$ is called a partially ordered set or *Poset.* We use strict partial orders to formalize the orders established between abstraction levels and between wps(s) with respect to the *implements* traceability relation.

9.5 The Implementation Relation and Levels of Abstraction

In this section, we analyze and formalize the implementation relation between wps(s) and their corresponding associated levels of abstraction.

9.5.1 The Need for Analyzing the Semantics of the Implementation Relation

After conducting structured interviews with requirements traceability users and analyzing leading Computer Assisted Software Engineering tools (CASE) for requirements management, Ramesh and Jarke [219] justify the need for analyzing the semantics of traceability relationships.

> An important concern of the study participants was the lack of support in many CASE tools for the automated identification of derived links. ("I do not have the time to link every requirement with everything produced at different stages. These links must be automatically derived whenever possible.") For example, requirements may be linked to designs that are intended to satisfy them. Designs, in turn, may be linked to relevant system components. Then, a derived link is created from requirements to system components. Mechanisms for automated inferencing incorporated in deductive or active database environments such as ConceptBase can be used to infer "derived links" in a traceability environment. As a critical first step, the semantics of the different linkages between objects must be clearly represented.

This section directly addresses this "critical first step" [219] toward mechanically discovering these linkages by formalizing the semantics of the partial implementation relationship (*implements*) between wps(s).

Although the formal model described in Section 9.4 was designed to accommodate the modeling of any traceability relationship, in this article, we are concerned with analyzing and modeling the implementation relationship. The semantics of other types of traceability relationships need to be further analyzed and described in order to be included in our formal model.

9.5.2 The Partial Implementation Relation

Informally, *partial implementation* between two wps(s) indicates that a wps s_i is a more refined/detailed description of another wps s_j. In order to formalize this traceability relationship we introduce the *implements* binary predicate and define *partial implementation* between a wps s_i

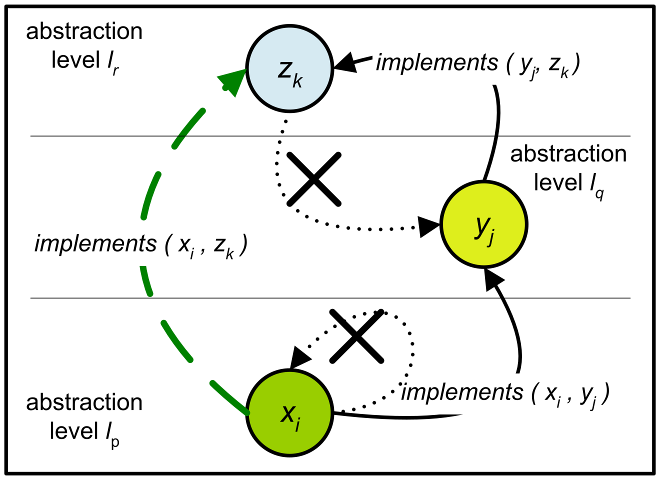

Figure 9.1: The partial implementation relation formalized as a strict partial order between work product sections.

and a wps s_j as an axiom in Γ of the form $implements(s_i, s_j)$. Now, we would like to model the *structural* properties of this *implements* relation with a well-studied mathematical formalism from which we can infer or enforce desired properties (formalized as inference rules in Γ). We would like to answer questions such as: Can we model the *implements* relation with a graph? What about modeling the *implements* relation using a tree, partial order, semi-lattice, topological space, or matroid? We analyzed these mathematical structures and believe that the mathematical model which best fits the current semantics of the implements relation is a strict partial order. Research work on studying the advantages and disadvantages of enforcing a lattice structure to the *implements* traceability relation is underway.

This leads us to our first group of inference rules for our Γ system:

Inference Rule 1. ***Implements Irreflexive Property.*** *A wps x_i does not partially implement itself.*

Inference Rule 2. ***Implements Asymmetric Property.*** *If a wps x_i partially implements another wps z_j, then z_j cannot partially implement x_i.*

Inference Rule 3. ***Implements Transitive Property.*** *A partial implementation traceability relation (implements) is transitive.*

9.5.3 The Implements Relation and Levels of Abstraction

From an implementation point of view, a system development project can be viewed as the composition of two major phases: 1) *Analysis* and 2) *Synthesis*.

The analysis phase is usually divided into smaller stages, e.g., *System Requirements Analysis*, *System Architecture Definition*, and *Software Requirements Analysis*. In each of these stages, starting from an initial system requirement or intention, the system is incrementally divided into components that partially implement a wps of a previous stage. The last stages in the analysis phase are the construction of hardware components and the development of code modules.

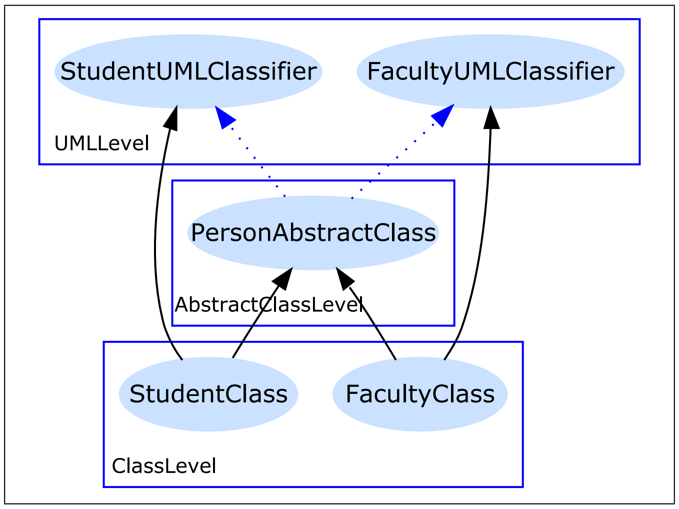

Figure 9.2: Example of levels of abstraction and of dependencies uncovered by the conceptual completeness principle. Traceability relations key: *implements*: solid (black) arrows, *implements*(transitive): dashed (green) arrows (not present in this figure), *impDependency*: dotted (blue) arrows.

In the synthesis phase, the system is incrementally composed from its parts starting with the code modules and hardware components by compiling, linking, assembling, deploying, and configuration processes.

Each refinement step in the analysis phase and each composition step in the synthesis phase inserts a set of newly created wps(s) associated with a given level of abstraction, which can also be called level of refinement. If such an abstraction level is already existent, then the newly created wps(s) can be associated with (linked to) it. Otherwise, a new level of abstraction is created and the new wps(s) linked to it.

In order to model these relationships: i) between abstraction levels and ii) between abstraction levels and wps(s), we will introduce, in Γ, two new binary predicates: *higherAbstractionLevel* and *abstractionLevel*.

Strict Partial Order between Abstraction Levels: In Γ, we construct a strict partial order between the abstraction levels by using a formal traceability link of the form *higherAbstractionLevel* (l_p, l_q), which indicates that the abstraction level l_q is of a higher abstraction level (more abstract and less refined) than the abstraction level l_p.

A simple *Student-Faculty* example developed for the purposes of describing this concept (and conceptual completeness) is presented in Figure 9.2. In this example, there are three levels of abstraction and their ordering is as follows: *UMLLevel* > *AbstractClassLevel* > *ClassLevel*, where *UMLLevel* is the most abstract and *ClassLevel* is the least abstract.

In Figure 9.7 we show a diagram of the strict partial order between abstraction levels we have established for our Guidance and Control Software (GCS) case study, which we describe in Section 9.8. In Figures 9.2 and 9.4–9.6, levels of abstraction have been represented by a frame grouping their corresponding associated wps(s).

Work Product Sections of an Abstraction Level: In Γ, and in order to indicate that a given wps s_i belongs to a certain level of abstraction l_q, we create a formal traceability link *abstractionLevel* (s_i, l_q). There is one and only one such link for each wps s_i.

In addition, wps(s) that are linked by a partial implementation traceability link necessarily belong to different abstraction levels. Furthermore, given a traceability link of the form *implements*(s_i, s_j), wps s_i belongs to a lower level of abstraction than wps s_j.

Now we can introduce, in Γ, our second group of inference rules, which correspond to the formalization of abstraction levels and their relationship with wps(s).

Inference Rules numbered 4–6: **Strict Partial Order of Abstraction Levels.** These three inference rules correspond simply to the irreflexive, asymmetric, and transitive properties for the strict partial order imposed on the set $\mathbb{A}$ of abstraction levels.

Inference Rule 7. ***Existence and Uniqueness of an Abstraction Level for a given wps.*** *There exists one and only one abstractionLevel traceability link for each wps* $x_i \in \mathbb{S}$.

Inference Rule 8. ***The Strict Partial Order of Implementation for wps(s) Is Constrained to the Strict Partial Order of Abstraction Levels.*** *An implements traceability link from wps x_i to wps y_j can only exist if wps y_j belongs to a higher level of abstraction than wps x_i.*

9.5.4 Implementation Meta-Work Products

We define a new structure called a **meta-work product**. A meta-work product is a subset M of wps(s) y_j of $\mathbb{S}$, such that every y_j of M is linked, within a traceability relationship t, to or from a common wps x_i, and that every $y_j \in M$ belongs to the same level of abstraction.

For example, given the *implements* traceability relation and two comparable abstraction levels *source code modules* < *low-level design*: a meta-work product is the group of source code modules that collaborate to implement a specific low-level design component.

Definition 24. *Given a wps $y_j \in \mathbb{S}$, the* ***implementation meta-work product*** $M^{\alpha_r}_{implements}(y_j)$ *is the subset of wps(s) that belong to the same level of abstraction α_r, such that for every wps $z_k \in M^{\alpha_r}_{implements}$, there exists a traceability link implements (y_j, z_k) with source in y_j and target in z_k (or inversely), where α_r is a given level of abstraction. In other words, $M^{\alpha_r}_{implements}(y_j)$ is the set of components at a certain level of abstraction α_r that a given component y_j partially implements.*

$$M^{\alpha_r}_{implements}(y_j) = \{\ z_k \in \mathbb{S} \mid (9.6)\ \}\,. \tag{9.5}$$

$$\begin{aligned} \Gamma \vDash\ & implements\,(y_j, z_k)\ \vee \\ & \Gamma \vDash implements\,(z_k, y_j)\ ) \\ \wedge\, \Gamma \vDash\ & abstractionLevel\,(z_k, \alpha_r)\,. \end{aligned} \tag{9.6}$$

$$\begin{aligned} Where:\ & y_j \in \mathbb{S}\,,\ z_k \in \mathbb{S} \\ & and\, \forall z_k \in M^{\alpha_r}_{implements}(y_j) : z_k \neq y_j\,. \end{aligned}$$

Implementation meta-work products offer a formal clustering technique based on the entities of a system model defined by its levels of abstraction and the *implements* traceability relation.

9.5.5 Enforcing Partial Order Properties for Implementation Traceability Links

Our experimental system, SyModEx, can mechanically enforce and/or verify partial order properties for any desired linking relationship type. Based on an original set of links, which we call **given**, we enforce and verify a strict partial order for the implements relation between wps(s). We also verify the strict partial order established between abstraction levels. If a pair of symmetric traceability links is found, then the implements relation would be inconsistent with the strict partial order properties, and human intervention would be necessary in order to solve the inconsistency by removing one of the conflicting traceability links. Nentwich, Emmerich, and Finkelstein provide methods and a tool for managing inconsistencies in the frame of their *Xlinkit* approach [204]. Also, in previous work by Finkelstein et al., the authors describe the use of meta-rules, based on temporal logic, that indicate what to do in case inconsistencies between wps(s) are found [107].

9.6 The Conceptual Completeness Principle

In this section, we define what we call the **conceptual completeness principle**, which is a mechanical dependency derivation rule. This rule, when applied in combination with the inference rules introduced previously, allows us to determine and derive implicit implementation dependencies between wps(s) based on a set of explicitly *given* implementation traceability links and the order established for abstraction levels. This principle exposes dependencies that are not found using just transitivity because of the directional nature of the partial implementation relation. These hidden dependencies will be formalized in our formal model Γ using the predicate *impDependency* described in this section.

9.6.1 Definition of Conceptual Completeness

Assume that x_i, y_j, and z_k are wps(s) of $\mathbb{S}$ and that x_i, y_j, and z_k belong to three different levels of abstraction where z_k belongs to the most abstract level and x_i belongs to the least abstract level (Eqn. (9.9) and Figure 9.1).

Definition 25. *Given a formal model* Γ*, the conceptual completeness principle is defined by Equations* (9.7) *and* (9.8)*.*

$$\begin{aligned} &\forall x_i, \forall y_j, \forall z_k \in \mathbb{S} \\ &(\Gamma \vDash implements(x_i, z_k) \\ &\wedge \Gamma \vDash implements(x_i, y_j)) \rightarrow \\ &\Gamma \vDash impDependency(y_j, z_k). \end{aligned} \tag{9.7}$$

$$\begin{aligned} &\forall x_i, \forall y_j, \forall z_k \in \mathbb{S} \\ &(\Gamma \vDash implements(x_i, z_k) \\ &\wedge \Gamma \vDash implements(y_j, z_k)) \rightarrow \\ &\Gamma \vDash impDependency(x_i, y_j). \end{aligned} \tag{9.8}$$

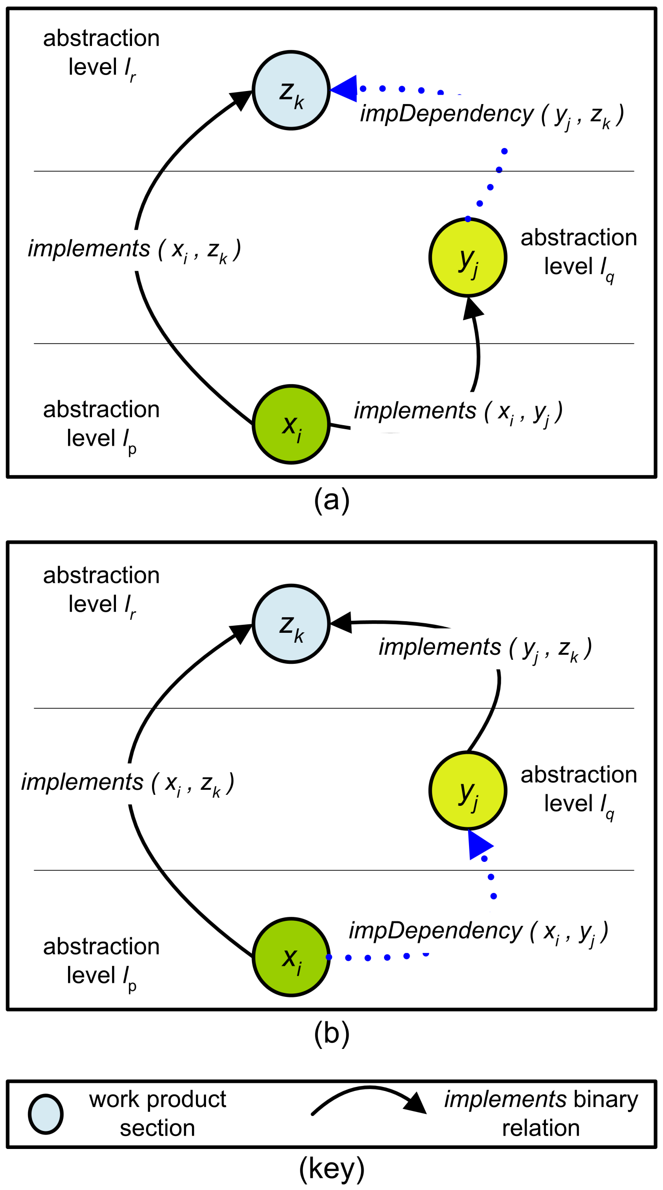
abstraction level l_r
z_k
impDependency (y_j , z_k)
implements (x_i , z_k)
y_j
abstraction level l_q
abstraction level l_p
x_i
implements (x_i , y_j)
(a)
abstraction level l_r
z_k
implements (y_j , z_k)
implements (x_i , z_k)
y_j
abstraction level l_q
impDependency (x_i , y_j)
abstraction level l_p
x_i
(b)
work product section
implements binary relation
(key)

Figure 9.3: The conceptual completeness principle.

$$\begin{aligned} Where:\ & abstractionLevel(z_k, \alpha_r) \wedge \\ & abstractionLevel(y_j, \alpha_q) \wedge \\ & abstractionLevel(x_i, \alpha_p) \wedge \\ & higherAbstractionLevel(\alpha_r, \alpha_q) \wedge \\ & higherAbstractionLevel(\alpha_q, \alpha_p). \end{aligned} \tag{9.9}$$

The rationale behind this rule is twofold. If wps x_i partially implements wps z_k, and x_i also partially implements y_j, then there is an implementation dependency from y_j to z_k (Eqn. (9.7) and Figure 9.3(a)); we call this *upward* conceptual completeness. And, if wps x_i partially implements z_k, and y_j partially implements z_k, then there is an implementation dependency from x_i to y_j (Eqn. (9.8) and Figure 9.3(b)); we call this *downward* conceptual completeness.

Figure 9.3 shows a graphical representation of this rule, and Figure 9.2 illustrates the rule using the *Student-Faculty* example. In this example, the two traceability links, represented by dotted arrows, *impDependency* (*PersonAbstractClass*, *FacultyUMLClassifier*) and *impDependency* (*PersonAbstractClass*, *StudentUMLClassifier*) indicate that there are two implementation dependencies which have not been explicitly specified as partial implementation links and cannot be found just with transitivity, but were discovered by conceptual completeness.

Now we define our third and last group of inference rules for Γ.

Inference Rule 9. ***Upward Implementation Dependency between wps(s).*** *Given three different wps(s) x_i, y_j, z_k belonging to three different abstraction levels, as described by Eqn.* (9.9), *then Eqn.* (9.7) *holds for x_i, y_j, z_k.*

Inference Rule 10. ***Downward Implementation Dependency between wps(s).*** *Given three different wps(s) x_i, y_j, z_k belonging to three different abstraction levels, as described by Eqn.* (9.9), *then Eqn.* (9.8) *holds for x_i, y_j, z_k.*

9.7 Case Study One: The Minimum Safe Altitude Warning System and the Guam International Airport Accident in 1997.

In this section, we describe a catastrophic aircraft accident at the Guam International Airport in 1997 which was related to the Minimum Safe Altitude Warning (MSAW) system. We develop a partial formal implementation model for the MSAW system and apply our technique for the discovery of hidden dependencies to the formal implementation model.

We only briefly introduce the reader to the accident and its probable causes and refer the reader to the corresponding NTSB (National Transportation Safety Board) report [200] and Knight et al.'s article [146] for further information on the accident and the MSAW system.

9.7.1 The Guam International Airport Accident in 1997 and the Minimum Safe Altitude Warning System

On August 6, 1997 Korean Air Flight 801 crashed at Nimitz Hill, Agana, Guam, about three miles southwest of the runway. The aircraft was completely destroyed by the impact and following fire. A total of 228 persons died as a consequence of this accident, and there were 26 survivors [200].

One of the causes that hindered the pilots and air traffic controllers from preventing this accident was an unsafe configuration setting in the Minimum Safe Altitude Warning (MSAW) system. MSAW is a ground-based system designed to provide aural and visual alerts to air traffic controllers (ATCs) if an aircraft has descended, or is predicted to descend, below a prescribed minimum safe altitude. Unfortunately, due to this misconfiguration, the MSAW system warnings were effectively only activated for a *one* nautical mile (nm)-wide ring, between 54 nm and 55 nm from the radar site. This misconfiguration hampered the air traffic controllers in service from receiving timely ground proximity warnings from the MSAW system to be forwarded to the pilots of Flight 801 [200].

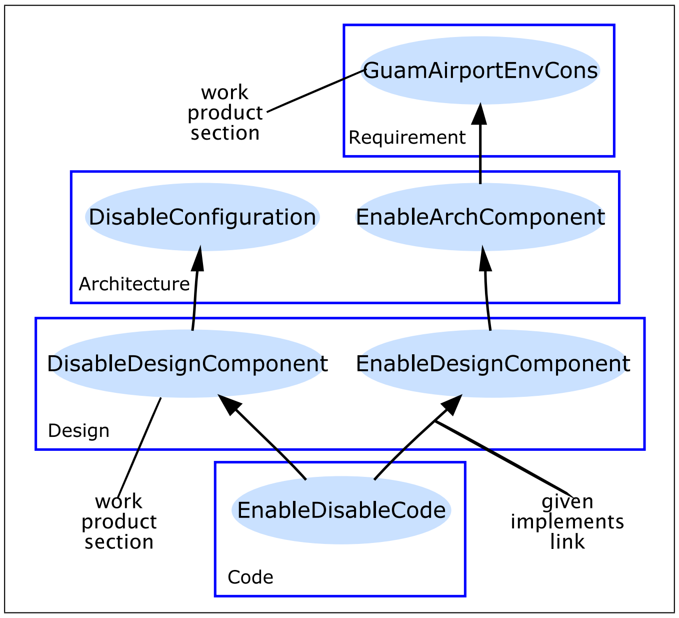

Figure 9.4: Formal implementation model of the *GuamAirportEnvCons* environmental constraint (MSAW case study). KEY: *implements* traceability link: solid (black) arrows; Work product sections: Ellipses (gray or light blue); Abstraction levels: frames enclosing wps(s).

9.7.2 Abstraction Levels and Formal Implementation Model for the MSAW Case Study

Figure 9.4 shows a formal implementation model created to represent this unsafe MSAW system inhibition configuration. In it, we represent the 54 nm MSAW system disable (inhibition) setting as an operational configuration parameter, labeled *DisableConfiguration*. We also represent the 55 nm MSAW system enable setting as an environmental constraint and label it *GuamAirportEnvCons*. The defined abstraction levels and their order are as follows: *Requirement* > *Architecture* > *Design* > *Code*.

The values of enable at 55 nm and disable at 54 nm were the reason the MSAW system for approaching aircraft was not able to generate adequate ground proximity warnings the night of the accident [200]. Notice that in this model (Figure 9.4) the *GuamAirportEnvCons* and the *DisableConfiguration* wps(s) appear to be **independent**.

9.7.3 Partial Order and Conceptual Completeness Applied to the MSAW Case Study

Airplane altitudes and descent trajectories are compared by the MSAW system against terrain altitudes and other information specific to each airport, which are specified in a configuration file [146]. This information is organized in a series of tables, typically on the order of 200 tables. The total number of lines in this configuration file, for a typical airport (as described in [146]) is about 25,000 lines, including 18,000 comment lines. The configuration table corresponding to the terrain map is the largest table with about 7,000 lines [146].

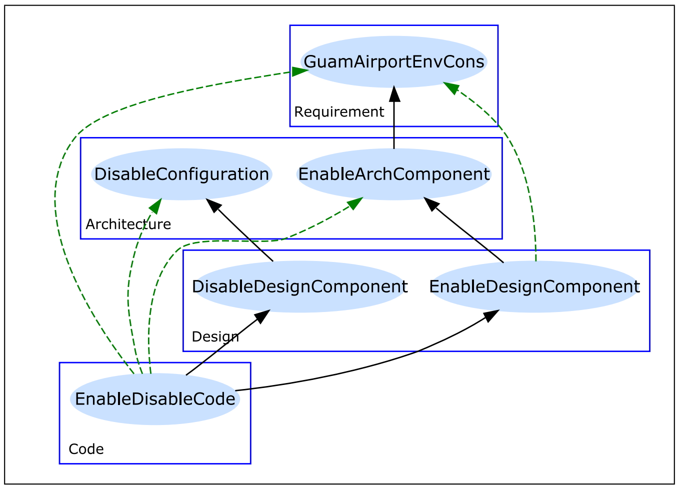

Figure 9.5: Formal implementation model of the *GuamAirportEnvCons* environmental constraint (MSAW case study) after enforcing strict partial order properties. KEY: *implements* traceability link: solid (black) arrows; *implements* (transitive) traceability link: dashed (green) arrows.

Notice that the size and complexity of this configuration file would preclude most human beings from noticing the dependency between the Guam Airport environmental constraint (*GuamAirportEnvCons* in Figure 9.4) of 55 nm for the MSAW system original operation and the disable configuration parameter set to 54 nm (*DisableConfiguration* in Figure 9.4).

Two quality assurance audits were conducted by FAA personnel of the Guam CERAP facility before the accident, one in 1995, which noted the inhibition only as an "informational" item, and a second one in 1997 (in the same year of the accident) which did not notice the safety hazard imposed by the ARTS IIA MSAW system inhibition settings. Neither of the evaluations indicated the inhibit configuration setting at 54 nm as a safety hazard [200].

We believe that had the air traffic controllers and the FAA auditors visualized this hidden dependency while configuring the system and/or while performing the audits, respectively, they would not have adopted nor kept this unsafe configuration.

Figure 9.5 shows the Minimum Safe Altitude Warning (MSAW) system case study formal implementation model derived from Figure 9.4 by adding all the links generated by the transitive closure for the *implements* traceability relation. The newly added links are shown in the graph using dashed arrows. Originally, there were five *given* links between wps(s). Figure 9.5 now shows four newly added links which were **derived by transitivity**.

Figure 9.6 shows the formal implementation model for the MSAW case study enhanced with four new links derived by conceptual completeness. The conceptual completeness principle was applied to this model after enforcing the strict partial order properties (as shown in Figure 9.5).

Notice that in Figure 9.6, in the *Design* abstraction level, the displayed order of the *EnableDesignComponent* and *DisableDesignComponent* wps(s) has been switched with respect to Figure 9.5. Recall that these figures have been mechanically generated by SyModEx and rendered by Graphviz. The switching of wps(s) in the drawing is due to the fact that the Graphviz layout algorithm we used, called *dot*, attempts, whenever possible, to avoid crossings between arrows.

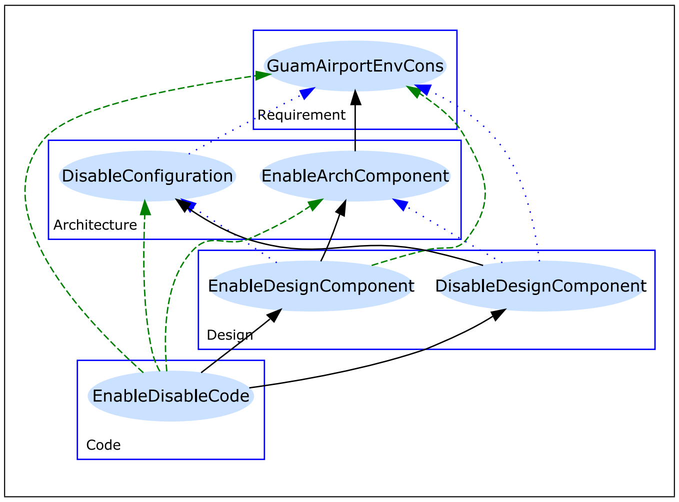

Figure 9.6: Formal implementation model of the *GuamAirportEnvCons* environmental constraint (MSAW case study) after enforcing partial order and conceptual completeness properties. KEY: *implements* traceability links: solid (black) arrows, *implements* (transitive): dashed (green) arrows, *impDependency*: dotted (blue) arrows.

The four new **implementation dependency links** are represented by the dotted (blue) arrows, and their formal representation is as shown below, where *impDep* stands for *impDependency*, and the words *Component* and *Configuration* were abbreviated as *Comp.* and *Conf.*, respectively.

$$impDep\,(EnableDesignComp, DisableConf)$$
$$impDep\,(DisableDesignComp, EnableArchComp)$$
$$impDep\,(DisableDesignComp, GuamAiportEnvCons)$$
$$impDep\,(DisableConf, GuamAiportEnvCons)$$

The enhanced formal implementation model shown in Figure 9.6 now indicates that the Guam Airport environmental constraint (*GuamAirpotEnvCons*) wps is, in fact, dependent on the *DisableConfiguration* architectural wps. This dependency was hidden in both the original model (Figure 9.4) and the model after enforcing transitivity (Figure 9.5).

After the application of our technique, this formal implementation model indicates that any modification to the *DisableConfiguration* wps needs necessarily to account for the environmental constraint *GuamAirpotEnvCons* wps.

9.8 Case Study Two: The Guidance and Control System

In this section we introduce the Guidance and Control Software (GCS) project, describe a model of the abstraction levels used for developing this project, and describe the formal implementation model corresponding to the Guidance and Control Software system lifecycle data. We also describe how the application of our technique helps discover hidden implementation dependencies and point out a potential failure mode associated with a hypothetical request for changes to the Temperature Sensor Processing routine.

9.8.1 Introduction to the Guidance and Control Software Project

The *Software Considerations in Airborne Systems and Equipment Certification* recommendation (1992) [228, 4], commonly known as DO-178B, was developed by the RTCA[3].

DO-178B provides guidance for the development and certification of software used in airborne systems. The same recommendation was adopted by EUROCAE[4] as ED-12B. The DO-178B/ED-12B procedures are normally used in conjunction with other safety standards in order to achieve a Federal Aviation Administration (U.S.A.) and Joint Aviation Authorities (Europe) (FAA/JAA) airborne system or component certification [255].

In 1995, the National Aeronautics and Space Administration (NASA) Langley Research Center (U.S.A.) prepared a set of lifecycle documents (including software) following the DO-178B/ED-12B recommendations for the FAA. These documents, which we will refer to as the *Guidance and Control Software (GCS) project lifecycle data*, were prepared only and specifically for use of the FAA at the 1995 Software Standardization Workshop held in Denver, Colorado, U.S.A., in August 1995. The *GCS project lifecycle data* are not commercial nor fully vetted products.

The GCS project lifecycle data reflect a DO-178B-based development process and deliverables of the Guidance and Control Software (GCS) for a **hypothetical** lander spacecraft module similar to the NASA Viking Lander. For example, some of the documents in the GCS project lifecycle data are *Software Requirements Data*, *Software Design Description*, and *Software Quality Assurance Records*. We developed our GCS case study based on these lifecycle data.

9.8.2 Abstraction Levels for the GCS Case Study

Based on the *Software Requirements Data*, *Software Design Description*, and *Software Source Code* GCS lifecycle data deliverables, we inferred the levels of abstraction that were implicitly used in the specification and development of the GCS project. Figure 9.7 shows a graph generated by SyModEx and drawn by the SHriMP visualization tool of the GCS abstraction levels and the partial order established between them.

In order to place the GCS control software in its hypothetical work environment, we added the following abstraction levels: *MainSystemRequirement*, *HardwareRequirements*, *Sensors*, *Actuators*, *HardwareImplementation*, and *SystemImplementation* (Figure 9.7). The other abstraction levels are a direct representation of the structure of the GCS software specification.

The links connecting abstraction levels (Figure 9.7) correspond to the *higherAbstractionLevel*

[3]The RTCA Inc. [14] was originally created as the Radio Technical Commission for Aeronautics in 1935. Today, known as RTCA, it is a private and not-for-profit corporation grouping more than 300 organizations worldwide. The purpose of the RTCA is to develop recommendations for the aviation industry concerning all aspects of aviation equipment [14].

[4]The European Organisation for Civil Aviation Equipment (EUROCAE) was formed in 1963 as a private forum for discussing technical aviation issues. Today, EUROCAE recommendations are used as the basis for certification of electronic systems and software for use in civil aviation in Europe [8].

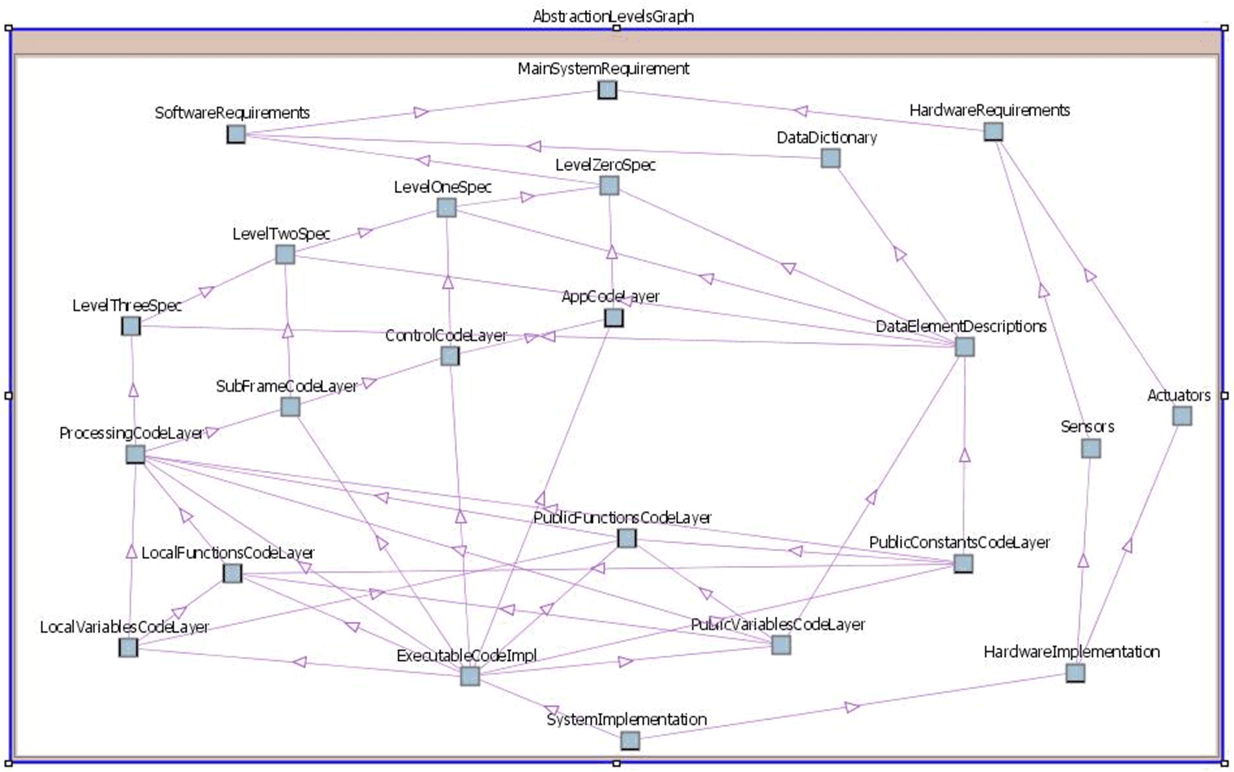

Figure 9.7: Partial order of the levels of abstraction defined for the Guidance and Control Software project as shown by the SHriMP tool (GCS case study).

relation and establish a strict partial order; transitive links are not shown in the figure. This order, as stated by the conceptual completeness principle inference rules in Γ, acts as a meta-model of dependencies between work products sections. In other words, the nonexistence of a formal *higherAbstractionLevel* link between any two given abstraction levels necessarily indicates that we can prove independence (or rigorously argue independence) between any two wps(s) belonging to each of these two abstraction levels. This is the result of the restriction introduced in Eqn. (9.9), which allows us to constrain, based on assumptions about the system and its development process, the number of dependencies derived by conceptual completeness.

For example, in most systems, we could assume that the *Sensors* and *Actuators* abstraction levels are independent from each other, and in such a case both abstraction levels will not be comparable with each other in the formal model (i.e., no *higherAbstractionLevel* traceability link between them). In the same case, an *implements* traceability link from a *Sensor-S1* wps to an *Actuator-A1* wps will be inconsistent with Inference Rule 8. Furthermore, due to the restriction established by Eqn. (9.9), the completeness principle will not generate an *impDependency* link between *Sensor-S1* and *Actuator-A1*. Again, such assumptions of independence must be rigorously justified, or the abstraction levels in question must be considered partially ordered with respect to each other in the formal model.

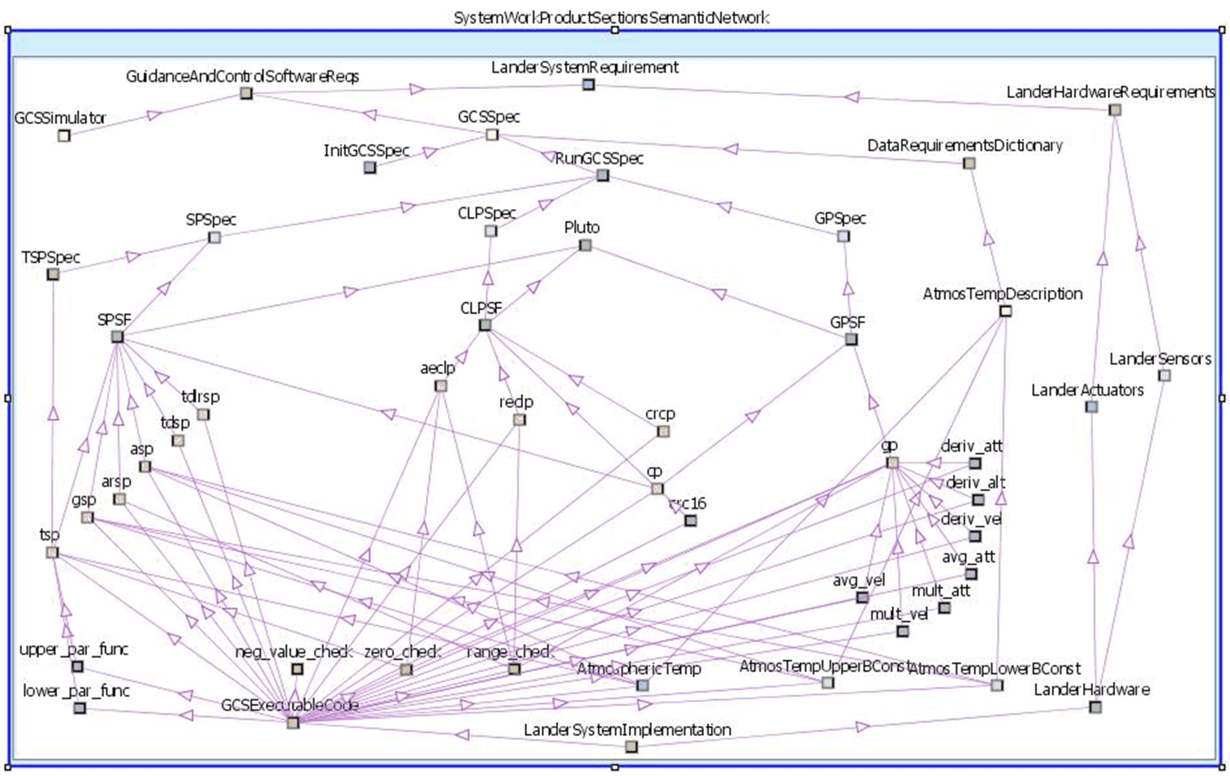

Figure 9.8: Formal implementation model of the Guidance and Control System (GCS case study).

9.8.3 Formal Implementation Model for the GCS Case Study

Based on the *Software Requirements Data*, *Software Design Description*, and *Source Code* documents of the GCS lifecycle data, we created a formal implementation model for the GCS project wps(s) (Figure 9.8).

Based on the call-tree of the source code, plus the information in the *Software Requirements Data* and *Software Design Description* documents, we inferred the *implements* relation between the GCS project wps(s). In a framework where Assumptions I, II, and II are valid, this procedure could be mechanically performed by tools.

In Figure 9.8, at the *LevelThreeSpec* abstraction level, only the wps named *TSPSpec* (Temperature Sensor Processing specification) was included in the graph, in order to improve clarity of the model. This should not be considered, in any way, a limitation; rather the SHriMP artifact navigation tool has been designed with the functionality of node filtering, as well as connector filtering, which allow stakeholders to navigate and visualize only those wps(s), or links, of current interest.

9.8.4 The GCS Temperature Sensor Processing Routine

In the GCS lifecycle data *Software Requirements Data* document, the Temperature Sensor Processing specification (*TSPSpec* in Figure 9.8) describes the purpose of, and the process for, calculating the atmospheric temperature. The atmospheric temperature is used by other processes to adjust the response of the gyroscopes and accelerometers. This *TSPSpec* wps is implemented by the *tsp* source code routine (also shown in Figure 9.8).

Two hardware sensors, a thermocouple pair and a solid state sensor, provide raw bit data to the *tsp* routine by means of the *THERMO-TEMP* and *SS-TEMP* inputs, respectively. Based on these two raw bit input values, the *tsp* routine calculates the atmospheric temperature and saves the output into the *ATMOSPHERIC-TEMP* public variable, represented in Figure 9.8 by wps *AtmosphericTemp*. It also sets a *TS-STATUS* variable to a value recognized as *healthy* or *faulty*. The thermocouple pair is more accurate than the solid state sensor, but its range of measurement is smaller. The task of the *tsp* routine is to decide which sensor to use, convert the raw bit values into temperature values, and store the result in *ATMOSPHERIC-TEMP*.

9.8.5 A Hypothetical Request for Changes to the Temperature Sensor Processing Routine

Assume that a change to the Temperature Sensor Processing routine is requested as part of an organizational move to the International System of Units. Temperature values previously expressed in degrees Celsius now need to be changed to Kelvin.

We start by searching for the part of the specification where the temperature is calculated. By following the *Software Requirements Data* document, we find the *TSPSpec* wps as the specification of concern and, following the *given* traceability links shown in Figure 9.8, we know that the routine called *tsp* implements the *TSPSpec* wps. We analyze both *TSPSpec* and *tsp* and verify that *tsp* is effectively a direct implementation of the process described in *TSPSpec*.

The *tsp* routine uses linear and parabolic functions to calculate the temperature based on the raw bit inputs. The slope of the linear functions is determined by using four parameters for each input, which specify two points in a planar space. Those parameters are: in the solid state case *T1, M1, T2*, and *M2*, and in the thermocouple sensor case *T3, M3, T4*, and *M4*.

We observe that there are no direct changes to make in the code of the *tsp* routine itself since all values have been parameterized. As a second step, we search in code for the definition of the parameters which have been directly used in the *tsp* routine. We find that the actual values for these parameters are not specified anywhere in the source code. These parameters are part of what are called *run parameters* and are set as part of the initial configuration of the GCS by an external application (the *GCS Simulator*).

Here we might wrongly assume that our modification job is concluded. The *TSPSpec* does not specify that the calculated atmospheric temperature needs to be checked for validity. Any value within the ranges of the previously indicated run parameters will be considered valid by the *tsp* routine, and we might wrongly assume that any value generated by the *tsp* routine is a valid temperature value.

The conflict resides in the fact that, after the requested change, the atmospheric temperature has now a range of validity which differs from the range specified for the run parameters *T1, T2* or *T3, T4* used to calculate the slopes. Recall that these run parameters have not been adjusted because we did not know about their dependency on the atmospheric temperature calculated by *tsp*. However, the range of the atmospheric temperature is checked before its use in the gyroscope sensor processing (*gsp*) and the accelerometer sensor processing (*asp*) routines. If the temperature

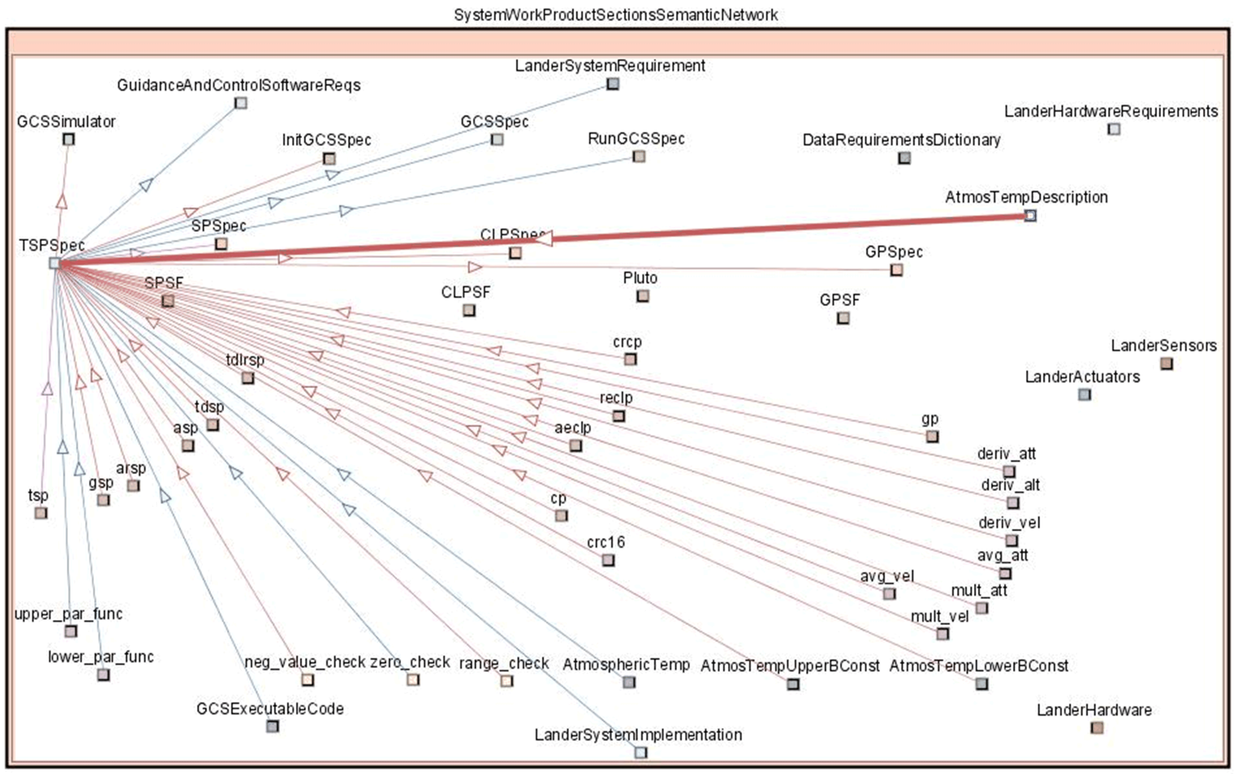

Figure 9.9: Dependencies to and from the *TSPSpec* wps (GCS case study). Observe the dependency with the *AtmosTempDescription* data dictionary entry. Only *impDependency* traceability links to and from *TSPSpec* are displayed.

falls outside the predefined valid range (-200 to 25), values which are represented in our formal model by wps *AtmosTempLowerBConst* and wps *AtmosTempUpperBConst* (Figure 9.8), the GCS control software will, in the case of a simulated environment, report an error. In the case of a real setting it may result in the loss of the mission.

There is no mention in the *TSPSpec* specification, nor in the *tsp* code or comments, that the valid ranges for the atmospheric temperature differ from those ranges indicated for the run parameters used to calculate the slopes. In neither *tsp* nor *TSPSpec* is there mention that the atmospheric temperature calculated by *tsp* is checked for range validity by other routines: *gsp* and *asp*. There is also no mention in either the Gyroscope Sensor Processing specification (*GSPSpec*) or the Accelerometer Sensor Processing specification (*ASPSpec*) that the temperature will be checked for range validity after its value has been determined and saved by *tsp*. The reason why critical variable values are checked for range validity only before their use is due to a design decision which has been stated elsewhere and has not been propagated to each of the affected specifications. With a change in units to the run parameters, the values of the boundaries corresponding to interrelated constants and variables in the code must be modified. Neither a formal model of *given* nor *transitive* partial implementation traceability links would make this dependency explicit. However, after the application of our methodology for the discovery of hidden dependencies, this dependency is explicitly shown and with only one navigational step.

Figure 9.9 shows a formal model of the GCS case study after applying the completeness principle.

In this model, only the *impDependency* links to and from the *TSPSpec* wps are shown; all the other links have been filtered out by the SHriMP visualization tool.

With this enhanced model in hand, which directly shows the dependencies to and from the *TSP-Spec* wps, we can directly observe that there is effectively a dependency between the range values established for the atmospheric temperature by *AtmosTempUpperBConst* and *AtmosTempLower-BConst* and the *TSPSpec* wps, even though this dependency is not clear from the specification or the source code. Observe the highlighted link from the *AtmosTempDescription* wps to *TSP-Spec*. Also observe (in Figure 9.9), a link from *AtmosTempUpperBConst* to *TSPSpec* and a link from *AtmosTempLowerBConst* to *TSPSpec*. The visualization of these dependencies would help stakeholders ensure that all affected wps(s) are examined when making this change.

9.9 Related Work

For the purposes of this related work section, we classify the domain of discovering traceability links and hidden dependencies into three areas depending upon the discovery approach: statistically-based, formal (logic-based), and others. Works based on these three approaches are described in the first three subsections of this section. In the fourth subsection we describe Leveson and colleagues' work on Intent Specifications and its use of traceability. In the fifth and last subsection we describe the relationship between our approach to discovering hidden dependencies and Formal Concept Analysis.

9.9.1 Statistical Approaches to the Discovery of Traceability Links

Information retrieval-based approaches [120] to the discovery of traceability links have received the most attention from the research community over the years. These techniques attempt to recover traceability links after assuming that the required traceability information is not present, not reliable, or outdated and needs to be recovered or reconstructed (e.g., legacy code and code without links to design or requirements).

At the Research Centre on Software Technology, University of Sannio, Benevento, Italy, Antoniol and colleagues have used statistically-based information retrieval methods for recovering traceability links between source code and designs, and also between source code and textual documentation [44, 49, 43, 50, 42, 46, 48, 47, 45, 51, 108].

Antoniol, Canfora, Casazza, De Lucia, and Merlo used vector space information retrieval and probabilistic approaches to recover traceability links from C++ and Java source code to user manuals and functional descriptions, respectively, based on the assumption that names of identifiers in the source are similar to those in other software documentation [44, 42, 46, 47].

Antoniol, Casazza, and Cimitile devised a Bayesian approach (using previous knowledge of traceability links) to discover and propose traceability links to an analyst for approval and applied the method to recover traceability links from Java source code to functional specifications [50].

Another application of statistically-based distance measures from the Research Centre on Software Technology has been the comparison of low-level designs extracted form C++ source code with corresponding, but not updated, software design documents in order to help analysts with the upkeep of design documents [48, 45, 51, 108].

In the Department of Computer Science at the University of Kentucky, Lexington, Kentucky, U.S.A., Huffman Hayes and colleagues have also been working on statistical approaches to traceability link discovery [129, 128].

Huffman Hayes, Dekhtyar, and Osborne compared three statistical information retrieval methods to recover traceability links from requirements to other wps(s). Their three methods are: vanilla vector retrieval, retrieval with key-phrases, and thesaurus retrieval. They found that the latter method outperforms other keyword-based methods and also outperforms, in terms of number of recovered links (*recall*), a senior analyst performing manual recovery [128].

In a later paper by Huffman Hayes, Dekhtyar, and Karthekeyan Sundaram, the authors amalgamate these statistical methods into a tool called RETRO (REquirements TRacing On-target) [129]. A similar tool is the *KAS* tool being developed at the Software Engineering Research Laboratory (SERL) [16].

At the DePaul Center for Applied Requirements Engineering, DePaul University, Chicago, Illinois, U.S.A., Cleland-Huang and colleagues are also using statistically-based information retrieval approaches for the discovery of traceability links [80, 79, 239].

Settimi et al. applied four text-based information-retrieval algorithms to recovering traceability links between requirements, UML models, and source code [239]. The four algorithms were: a vector space model method with and without the use of a thesaurus and a pivot normalization score method, a method based on the vector space method with an adjustment for document size, also with and without the use of a thesaurus.

The results of their analysis are dissimilar to those of Huffman Hayes and colleagues. They found that their thesaurus-based approaches did not improve the *recall* metric and diminished the *precision* metric. They also found that both pivot normalization methods under-performed with respect to their unadjusted vector space counterparts. The best results were obtained when recalling links from requirements to UML Use Case Diagrams, showing that statistically-based approaches are extremely sensitive to the way keywords are used. Another interesting result reported in their paper is that none of the algorithms performed well when recalling links from requirements to source code. This case is consistent with what we observed in both case studies introduced in the present article. Our methodology discovers this kind of hidden dependencies because it formally transports implementation dependencies across all levels of abstraction.

Other researchers have used information retrieval methods for the recovery of traceability links. For example, researchers at their respective Departments of Computer Science at Kent State University, Kent, Ohio, U.S.A. and Wayne State University, Detroit, Michigan, U.S.A. have used Latent Semantic Indexing [168, 178].

In order to address precision and recall problems encountered by most information retrieval methods [80], Cleland-Huang et al. developed three improvements that were included in a probabilistic algorithm for the retrieval of traceability links between requirements and UML Class Diagrams [80]. The three enhancements were developed by the authors after analyzing the main causes of imprecision and insufficient recall of the previously applied statistical search algorithms. The enhancements are called: hierarchical, clustering, and graph pruning. The authors applied the probabilistic network retrieval algorithm, alone and in combination with each of the three previously mentioned enhancement strategies, to three different data sets. Only for one of the three data sets, named IBS, the traceability link retrieval results improved for each and all of the three enhancements. After further analysis of the IBS data set, the authors say: "...the artifacts were structured into useful contextual groupings that provided contextual clues" [80].

Independently from the results achieved by these three enhancement strategies proposed by Cleland-Huang et al., which were mixed results, the strategies themselves stress the importance of our approach. For example, the case with the IBS data set is another positive case that verifies our hypothesis that the semantic structure of a system model is a very important clue to finding hidden dependencies. In fact, our formal abstraction levels may be considered a formalization of the hierarchical enhancement introduced by Cleland-Huang, et al., and our implementation meta-work products abstraction resembles a formalization of the clustering technique proposed by the same authors [80].

In our work, we assume there is a predefined set of implementation traceability links in place that analysts and engineers have defined *a priori*. If this predefined or initial set is not present, even though its presence is mandated by standards, or is insufficient, it could be created or enhanced by using these statistical techniques and their associated tools. Then, based on this enhanced set, our technique mechanically discovers other hidden relationships that these statistical techniques may not completely find. For this reason, statistically-based approaches and our formal approach are not opposite but complementary and should be used in combination when needed.

We believe that Cleland-Huang would agree with us in the assertion that these approaches should be used in combination. In a recent position paper, she describes what she calls Goal Centric Traceability (GCT) [77]. GCT is a framework targeted at improving traceability of nonfunctional requirements [77], which can be seen as requirements or constraints on emergent properties. In the paper, Cleland-Huang states that both formal and statistically-based techniques, which she calls explicit and dynamic, respectively, are to be used depending upon the criticality of the requirement being traced.

The issue of determining the criticality of wps(s) has also been a target of our interest. In a previous article, we described work on using the Extensible Markup Language (XML) [6] and the Extensible Stylesheet Language Transformations (XSLT) [3] in order to propagate criticality of requirements into other lower abstraction level wps(s) [85].

9.9.2 Other Formal Approaches to Traceability

Pinheiro and Goguen developed a tool for tracing requirements called TOOR [215, 116]. In their approach, as in ours, traceability relationships are treated as mathematical relations between what they call objects (referred to as wps(s) in this article). In both approaches, traceability relations are defined by axioms in a logic system. In the TOOR tool, axioms are implemented by modules written in the Functional and Object-Oriented Programming System (FOOPS) [69, 115]. In our SyModEx system axioms and theorems are implemented by Prolog predicates.

However, Pinheiro and Goguen's work on TOOR focused on developing a tool for requirements traceability. The work described in this article focuses on analyzing and defining the semantics of the partial implementation traceability relation and on devising a technique for the discovery of dependencies that have not been explicitly stated. Our partial order and completeness principles defined for the implements traceability relation may have been implemented as TOOR relations using FOOPS modules instead of our Prolog-based implementation. In fact, the TOOR tool allows for traceability relations to be defined as transitive by creating a FOOPS module. Nevertheless, TOOR does not recognize the presence of levels of abstraction and does not implement a completeness principle with the purpose of discovering hidden dependencies between wps(s) in a system model.

Another formal approach to traceability is the work of Nentwich, Emmerich, Finkelstein and colleagues at the University College London, London, United Kingdom [204, 205, 202, 203, 107]. Nentwich, Emmerich, and Finkelstein search for inconsistencies between wps(s) using a tool called *Xlinkit* [202]. Their work is also based on a predicate calculus and uses XML as a representation vehicle for work products. The difference with our approach is that their work focuses on pointing out inconsistencies between wps(s) and specifying how to act when inconsistencies are found, based on a set of meta-rules, which are represented using XML as well. The work described in this article focuses on mechanically completing traceability relations based on a set of rules. We assume that an initial set of links is present and consistent. In other words, the *Xlinkit* approach and ours are complementary; their approach focuses on consistency of traceability links and ours focuses on partial completeness.

9.9.3 Other Approaches to Traceability

An information management-oriented approach to traceability is the work of Anderson, Sherba, and Van Lepthien at the University of Colorado, Boulder, Colorado, U.S.A. [259, 241, 41]. These authors and their colleagues developed a framework and prototype tool, InfiniTe, for the discovery, creation, and management of traceability among software artifacts. Their approach is called *information integration* and is based on four processes: discovery, creation, maintenance, and evolution, along with supporting tools for the management of relationships between software artifacts. In their approach, they use *Integrators*. Integrators can create new traceability links by "chaining" existent links [41]. Sherba, Anderson, and Faisal describe a particular instance of an *Integrator* used to chain traceability links to a new derived link, which in fact corresponds to a transitive relation [241].

The differences between Anderson, Sherba, and Van Lepthien's approach and ours are similar to the differences between Pinheiro and Goguen's approach and ours. Anderson, Sherba, and Van Lepthien's approach focuses on the development of a framework and tools to support traceability; we focus on gaining a deep understanding of each traceability relation and on clearly defining the semantics behind them. Furthermore, the completeness principle developed for the discovery of hidden dependencies presented in this article is a completely new methodology. As with the TOOR tool and FOOPS modules, our partial order and completeness principle axioms may be implemented using Integrators for the InfiniTe prototype.

A different approach from Cleland-Huang and colleagues, which is based on the publish-subscribe paradigm, uses change events to notify related artifacts upon the occurrence of changes [78]. This approach is focused on managing the evolution and change of traceability links during the maintenance process and does not address the issue of mechanically generating new links to uncover hidden dependencies.

Yu and Rajlich define a hidden dependency as a relationship between two apparently independent object-oriented source code structures (e.g., classes, methods, and parameters) [276]. They use an abstract system dependence graph that represents the hierarchical structure of a class hierarchy (up to the method level) enhanced by data-flow dependencies. Henrard and Hainaut devised a similar approach but based on a variable dependency graph generated from COBOL source code [132]. These approaches are somewhat similar to our technique for the discovery of hidden dependencies. However, they are not formal and they are constrained to source code data-flow dependencies only.

Egyed and colleagues have developed a methodology for the discovery of traceability links which is based on source code signatures and is the closest to our approach that we have found in the literature. Egyed [99, 98] and Egyed and Grünbacher [100] report on the application of this approach to a UML-based Inter Library Loan case study (ILL). Egyed and colleagues' methodology and *TraceAnalyzer* tool start with a set of traceability links generated by source code profiling, the authors used Rational PureCoverage® (now IBM® Rational PurifyPlus®). The source code signatures (set of classes) were collected during the execution of a set of test scenarios. The set of profiled (*observed*) traceability links is then enriched with a series of manually *hypothezised* traceability links. The union of these two sets of links closely corresponds to our *given* set of formal traceability links.

After a series of graph transformations, which Egyed calls *Atomize, Normalize, Generalize, Refine, and Generalize Specific*, the TraceAnalyzer tool generates a set of dependencies based on the idea that two UML model elements are mutually dependent if they share a common source code footprint [99]. Egyed's rationale and ours are very similar but the methodology and results are different. For example, in page 129 [99] Egyed states that due to its conservative approach, TraceAnalayzer did not mark code elements {1, 3, 5, 7} as dependent with model element [c2]. After attempting to replicate Egyed's ILL case study traceability data, our technique marked as dependent to the [c2] model element the three code elements that were marked by Egyed ({0, 8, 9}) and code elements {1, 3, 5, 7} as well. This serves as verification that our technique makes the safe assumption that, in case of doubt, any two wps(s) need to be deemed dependent; which is, as demonstrated in our introduction, a must for HA&CCS.

In addition, our technique is more general and applicable to all wps(s) at all levels of abstraction. For example, our methodology accounts for the case when two related requirements transport their relationship downward to architecture and design wps(s) even without the presence of source code, which makes it applicable at any stage of system development. Furthermore, the methodology introduced in this article is better suited to the analysis of nonfunctional requirements and emergent properties, and recognizes and formalizes the existence of different levels of abstraction, as well as the existence of implementation meta-work products, as semantic structures of a formal implementation model.

9.9.4 The Intent Specifications and SpecTRM Approach

Leveson and colleagues at the Massachusetts Institute of Technology, Cambridge, Massachusetts, U.S.A., have developed a safety and human-centered approach to the design and specification of HA&CCS [152, 154, 153, 160, 195]. The methodology which has been named SpecTRM (Specification Toolkit and Requirements Methodology) can be considered composed of several techniques all embedded in a human-centered and safety-driven analysis and development process [152]. The foundation techniques are *Intent Specifications* and the *SpecTRM-RL* requirements specification language.

Intent Specifications is a methodology for the specification of system requirements which was developed with humans and safety in mind. An intent specification describes a system at five different levels of abstraction. In the first level (*System Purpose*), the goals, main requirement, and constraints of the system are described. Also in the first level, safety and environmental constraints are specified. In the second level (*System Design Principles*) the components and design principles of the system are specified. In the third level (*Black-Box Behavior*), the behavior of system components is specified using a black-box approach. In the fourth level (*Physical and Logical Function*), detailed design descriptions of software and hardware components are specified. The last level (*Physical Realization*) corresponds to the system implementation itself. Each entity in each level of an intent specification is uniquely identifiable and also linked for traceability to and from lower and higher specification level entities [154].

The five levels of an Intent Specification can be considered a meta-model for the levels of abstraction used in this article. In addition, the traceability links between entities in an intent specification can be considered partial implementation links in our approach.

SpecTRM-RL is a language for specifying the black-box behavior of subsystems or components with a high level of abstraction. SpecTRM-RL uses a graphical notation and AND/OR tables for specifying behaviors as state machines. SpecTRM-RL was designed with readability in mind and has been shown to be understandable by engineers with different backgrounds. Specifications represented in SpecTRM-RL are executable, and a wide range of techniques can be applied to them to help engineers with the development, verification, and safety analysis processes, for example: completeness and consistency analysis [153], simulation and animation, state machine hazard analysis, software deviation analysis, and fault tree analysis, among others [160, 195].

The SpecTRM approach and methodologies are being successfully applied to specify, among others, air traffic management systems [152, 160]. The SpecTRM toolkit is an implementation of a framework such as the one described in the introduction of this article where wps(s) are uniquely identifiable and traceable. Hence, our technique for the discovery of hidden dependencies could be included in the SpecTRM approach as another safety analysis tool.

9.9.5 Formal Concept Analysis and Our Conceptual Completeness Principle

In another track of our research endeavors, we are using, in combination with our set theory and predicate calculus approach, a mathematical formalism called Formal Concept Analysis (FCA) [113]. In fact, the presence of a meta-work product in a given abstraction level α_r will indicate the presence of an *intent* or an *extent*, in FCA nomenclature, after the application of the partial order and conceptual completeness principles and by considering the union of the *implements* and *impDependency* traceability relations. This *intent/extent* belongs to the formal context generated by the implements relation between the abstraction level α_r and any other comparable abstraction level. Such a relationship is what inspired us to name our technique the *conceptual completeness principle*.

Tilley, et al. surveyed the use of FCA for software engineering activities and reported that most of the research in this area focuses on the application of FCA to concept discovery in legacy systems and refactoring of class hierarchies in object-oriented code [258]. An emerging trend is the use of FCA for analyzing formal requirements. As an example, Tilley has used concept lattices to visualize formal specifications (requirements and design) written in the Z language [257, 256]. In our work, we take the application of FCA to systems development a step further. We use FCA to analyze the structural properties of a whole system model, from requirements to system implementation, as a network of wps(s) connected by their corresponding traceability links [82].

9.10 Conclusions and Scope

The enormous complexity of today's systems, with their associated growing number of disparate engineering disciplines needed, along with the usually vast amount of textual documentation and information involved in the development, maintenance, and operation of a system, precludes any engineer, manager, operator, and/or auditor (stakeholder of the system model) from having complete detailed knowledge of a system and its underlying model. Therefore, an effective and mechanized approach to traceability between wps(s) is imperative in order to offer stakeholders the ability to navigate through related/dependent wps(s).

The contributions of the work reported in this article are threefold. Firstly, we describe an integral framework which offers a systems-oriented and effective solution to the traceability problem and thus enables the analysis of emergent properties. Secondly, we offer a much needed analysis and a formal model for the semantics of the partial implementation traceability relation. Thirdly, we describe a formal technique for the discovery of hidden implementation dependencies between system components which could lead to critical or catastrophic errors if left undetected. We expand on the last two of these contributions below.

Even though the importance of traceability relationships has been remarked upon extensively by authors who have analyzed the problem (for example [219, 216, 119]), we have not found a formal analysis of the structural properties for any of the traceability relationship types described in the literature. In this article, we analyzed the implementation relation between wps(s) and formalized it as a partial order (Section 9.5). This analysis and formal model are the first steps toward formal descriptions of the semantics for all traceability relations. The need for such analysis was clearly stated by Ramesh and Jarke [219].

The practical benefits of formal descriptions and clear semantics, in addition to the benefits described in this article, are twofold. First, they will make possible an agreed nomenclature for traceability relations, which would also facilitate communication among practitioners and between industry and the research community. Second, such enhanced communication and agreement would facilitate the adoption of traceability tools by industry, as well as enable the inter-operation between traceability tools. We envision the development of a standard for the nomenclature and semantics of traceability relations with the consequent practical benefits to industry and the research and academic communities.

With respect to our discovery of hidden dependencies methodology, its main drawback is the occurrence of dependencies that may not present a safety risk (false positives). For example, in Figure 9.9, not all links imply the occurrence of a safety risk; each dependency needs to be analyzed in order to determine its implications for the safety, or possibly other dependability properties, of a system. Notice that such a case does not imply that the dependencies are not there, only that they may not indicate an unsafe interaction. We are currently working toward alleviating the problem of false positives based on the idea that after independence between wps(s) has been rigorously justified (or proved) it can be formalized in Γ as a formal traceability link using the unary negation operator and recorded as such. Further research is needed in this area to assess under which conditions argued (or proved) and recorded independences remain invariant to changes in the formal implementation model.

Furthermore, our approach is the only one known to the authors that makes the following safe assumption: *work product sections must be deemed related or dependent unless we can rigorously argue or formally prove their independence*. Given the nature of HA&CCS, and given that hidden or unknown dependencies and interactions have been shown to be a major cause of critical and catastrophic failures, we cannot afford to miss relations and dependencies that could lead to critical or catastrophic behavior.

We would like to remark that the conditions and events presented in the Minimum Safe Altitude Warning (MSAW) system case study were not, by any means, the only cause for the accident. As described by the National Transportation Safety Board [200] the MSAW-related aircraft accident at the Guam International Airport was the consequence of a series of interrelated conditions and events. These conditions and events contributed to the accident and hindered the Flight 801 pilots, air traffic controllers, and FAA auditors from detecting problems and thus from preventing the accident.

The formal framework and technique described in this article do not ensure that stakeholders will receive and/or clearly understand necessary information about a system model. There are risks associated with organizational structures and communication channels which could prevent essential information from reaching a stakeholder. Furthermore, assuming such information is available in a timely manner and that navigational aids are available, as described by Assumptions I, II, and III presented in the introduction of this article, there are still risks associated with stakeholders correctly understanding the information. These issues also need to be considered and addressed when designing methodologies for the specification, creation, management, and navigation of HA&CCS formal implementation models.

We do not believe that our approach should replace statistically-based information retrieval approaches but that both approaches should be used in combination if necessary. Information retrieval approaches are necessary to recover traceability links when no traceability information has been recorded or when the recording and management of such information is not economically justified. Notice that this is **not** the case for the high assurance and critical computing systems object of this study. We recognize, though, that if a portion of a system can be unequivocally argued and justified to be non-critical, then it could receive a different treatment with respect to its traceability requirements. The issue then becomes how to unequivocally separate the non-critical portions of a HA&CCS, for which we believe our approach may also be appropriate. This is an active track in our research agenda, but further work is needed in this area.

The formal model and technique for the discovery of hidden dependencies described in this article, lend leverage to the discovery of unsafe emergent behaviors by allowing stakeholders to directly observe, with only one navigational step, places where unexpected, and possibly unsafe, interactions between system components may occur.

We do believe that, under the assumptions made in the introduction, this formal approach and technique would have empowered engineers and stakeholders with information necessary to make better decisions with respect to the involved configuration value modifications described in the MSAW case study and the hypothetical modification described for the GCS case study.

Chapter 10

Security Policy Refinement and Enforcement for the Design of Multi-level Secure Systems

10.1 Introduction

In recent years, security has become a growing concern in software engineering research, especially software architecture research. As defined in the IEEE standards, security is one of the software system attributes that serve as quality or non-functional requirements (NFR). Other NFRs include performance, verification, acceptance, portability, reliability, maintainability, and safety [226]. NFRs refer to the whole software and thus cannot be presented in software architecture as components or functions offered by the system [35]. The overall software architecture and NFRs are closely related and should be studied together during architectural development. In the literature, much of the research focuses on how to satisfy NFRs such as reliability and performance [226, 60, 75, 90, 109, 225], instead of security. There has been some work on security architecture modelling [91, 196], but not on providing security architecture design guidance for architects. Banerjee, et al. discuss a software system's architecture and its trustworthiness (security, reliability, availability, fault-tolerance, and survivability) in general [59]. However, they provide no additional detailed guidance. Some researchers [235, 279] focus on how to satisfy security of computation at the level of programming languages, but not on satisfying security of complex systems at a more abstract level.

The notion of enforcing security policies at the architectural level is attractive because it allows security concerns to be recognized early in the development process and can be given sufficient attention in subsequent stages. By controlling system security during architectural refinement, we can decrease software production costs and speed up the time to market. This approach also enhances the role of the software architects by requiring that their decisions apply to functional decomposition, as well as to security fulfillment. As some researchers have pointed out [35, 60, 90] , effective refinement must be technology- and domain-specific. Also, Garlan [114] points out that refinement patterns must be explicit about what kinds of properties they are preserving in the refined design. In our work, the type of systems for which our design refinement patterns are valid are multi-level secure (MLS) systems. We focus specifically on achieving confidentiality for secure systems. In our previous paper [288], we presented a set of architectural refinement patterns. This paper extends the previous work by providing a policy refinement language to specify refinement rules for each pattern, as presented in Section 10.3, and proposes the hierarchy of refinement patterns in Section 10.4. This extension takes us one step further towards the formal verification of the proposed refinement patterns.

This paper is organized as follows: Section 10.2 introduces concept of MLS systems and software architecture; Section 10.3 proposes architectural refinement patterns that can be applied to design MLS systems, and a policy refinement language to specify the refinement rules of each pattern. Section 10.4 briefly discusses the hierarchy of patterns. An example of MLS system design is presented in Section 10.5 to illustrate the approach. Section 10.6 is the conclusion and future work.

10.2 Background

10.2.1 Multi-level Security and MILS Architecture

Traditionally, the model of a secure system includes the concept of multi-level security (MLS). Given a set of subjects, each with a clearance level, and a set of objects, each with a classification level, the idea behind the MLS concept is that the system will be processing objects at different classification levels, and the access to these objects is restricted, by security policy, to subjects with particular clearance. Classic security models, such as the Bell-LaPadula (BLP) model [62], have been used to specify the secure behavior of such MLS systems. The BLP model requires that information does not flow downward by imposing the following requirements.

The Simple Security Property. A subject is allowed a read access to an object only if the subject's clearance level is identical to or higher than the object's classification level.

The *-Property. A subject is allowed a write access to an object only if the subject's clearance level is identical to or lower than the object's classification level.

In this paper, we use security level to represent both the classification level and the clearance level.

The problem with full MLS systems is that they must be rigorously analyzed for security before they can be certified. Every portion of the MLS system must be analyzed to ensure that it properly handles labelled data and that there is no possible violation of the security policies. Even with a trusted computing base (TCB) architecture or reference monitor [40], there is often too much to evaluate.

The Multiple Independent Levels of Security/Safety (MILS) architecture was developed to resolve the difficulty of certification of MLS systems by separating out the security mechanisms and concerns into manageable components [33]. These components are classified based on the way they process data:

- **SLS** Single-Level Secure component that only processes data at one security level.

- **MSLS** Multiple Single-Level Secure component that processes data at multiple security levels but always maintains separation between classes of data. An MSLS process or device separates the data into independent streams with no communication between streams. A device that processes messages one at a time (such as an I/O device driver) may be such a device.
- **MLS** Multi-Level Secure component that deals with data at multiple security levels and transforms the data from one level to another according to internal MLS security policies of this component. Because of the potential seriousness of violating its security policies, an MLS component requires the highest level of scrutiny and verification. Typically this can be a device that will downgrade information from a higher level of security to a lower level through either filtering or the application of encryption technology.

A MILS system isolates processes into separate execution spaces, either physically or logically separated, which define a collection of data objects, code and system resources. These individual partitions can be evaluated separately, if the underlying separation architecture is implemented correctly. This divide-and-conquer approach will exponentially reduce the proof effort for secure systems. The MILS architecture is MLS if it enables the enforcement of the MLS security policies to regulate the communication between the applications and the resources, which are application specific and not part of the MILS architecture.

10.2.2 Software Architecture

Software architecture is usually described by components, connectors, and architectural configurations [190]. The following concepts are derived from Medvidovic and Taylor [190].

A *component* in an architecture is a unit of computation or a data store. It may be as small as a single procedure or as large as an entire application. Each component may require its own data or execution space, or it may share them with other components. The features of a component include interfaces, semantics, constraints, NFRs, etc.

A *connector* is an architectural building block used to model interactions between different components and rules that govern those interactions. It may be a message routing device, shared variable, buffer, dynamic data structure, client-server protocol, pipes, and so on. Connectors are characterized by their interfaces, semantics, constraints, NFRs, etc.

A *component's interface* is a set of interaction points between the component and the external world. It specifies the services (messages, operations, and variables) a component provides or that are required of other components in an architecture. In our approach, we augment the interface with security levels for each service.

A *connector's interface* is a set of interaction points between the connector and the components or other connectors attached to it. A connector does not perform any application-specific computations, and it exports as its interface those services it expects of its attached components. In our approach, we also augment the interface with security levels for each service.

*Architectural configuration*s are connected graphs of components and connectors that describe architectural structure. In this paper, we use directed graphs to represent the architectures throughout the refinement process.

10.3 Policy Refinement Language and Refinement Patterns

When designing secure systems, we want to avoid separately constructing abstract and concrete architectures and then proving that the concrete architecture enforces the security policies of the abstract one. Instead, the concrete architecture should enforce the security policies by construction, requiring no explicit proofs in its derivation. This can be accomplished through a series of small, local refinements, each of which involves the application of a refinement pattern. The local refinements are combined to form the larger composite concrete architecture, which is guaranteed to correctly implement the abstract architecture, meaning that the security policies are not violated by the concrete architecture. In this way, the architecture can be refined to be increasingly more concrete until the implementation architecture is achieved and the original security policies have been refined and are now enforced by the components and connectors of the concrete architecture.

In our paper, we do not focus on how to do functional requirements (FR) refinement, which has already been researched for many years with many approaches described in the literature (e.g., [197] and [214]). Instead, we focus on how to augment architectural refinement with security concerns. We present a set of refinement patterns for designing MLS systems to allow architects to decompose, aggregate, and eliminate components, connectors, and ports (interaction points of an interface). Each refinement pattern is represented by a pair of directed graphs representing the pre- and post-architecture of the refinement. In the graphs, a box represents a component, an arrow represents a connector, and a dot represents a port. The refinement rules that must be satisfied to make each refinement pattern correct are specified using a proposed policy refinement language.

A component/connector is said to be trusted if it is or will be designed to enforce security policies, while an untrusted component/connector enforces no security policies. The security policies enforced by components/connectors describe not what they should do (e.g., send some data), but how they should do it (e.g., send data embedded in legal CORBA messages and with correct security labels). In our work, a security policy is defined as a tuple p=($Intra$, $Inter$, $Domain$). The $Intra$ and $Inter$ define two sets of different types of rules of a security policy, a set of intra-level rules and a set of inter-level rules. Intra-level rules are rules that are not related to information flow between security levels. For example, "Messages from Database must be CORBA reply messages" is such an intra-level rule. This rule is not on cross-level information flow, but on message types. Inter-level rules are security rules that are related to cross-level information flows. For example, "A top-secret message cannot flow to a secret component unless it is downgraded" is such an inter-level rule. *Domain* is a set of services associated with ports.

An SLS policy is a policy that only has intra-level rules while the $Inter$ set is empty. An MSLS policy is a policy that may have intra-level rules, but must also have an inter-level rule that there is no cross-level information flow. An MLS policy is a policy that may have intra-level rules, but must also have inter-level rules to regulate cross-level information flows. A component/connector that enforces an SLS, MSLS, or MLS policy is called an SLS, MSLS, or MLS component/connector correspondingly. In the final concrete architecture, if a connector enforces security policies, it must be application independent.

Definition (Correct Refinement) During the design of an MLS system, the refinement of the architecture (components, connectors, ports, or a combination of these entities) is correct if and only if the security policies after the refinement do not violate the security policies before the refinement.

Ref	::=	Decomp {,RConds} $\mid$ Aggeg {,RConds}
Decomp	::=	*Product*(*policy_id*, plist) $\mid$ *Cascade*(*policy_id*, plist)
		$\mid$ *Feedback*(*policy_id*, plist) $\mid$ *ConnDec*(*policy_id*,plist)
Aggeg	::=	*ComAgg*(plist,*policy_id*)$\mid$ *ComConnAgg*(plist,*policy_id*)
plist	::=	*policy_id* {, *policy_id*}
Rconds	::=	Property $\mid$ Relation
Property	::=	pProperty(*policy_id*){,pProperty(*policy_id*)} $\mid$ *True*
Relation	::=	pRelation {, pRelation}
pProperty	::=	*isSLS* $\mid$ *isMSLS* $\mid$ *isMLS*
pRelation	::=	IntraR $\mid$ InterR $\mid$ SLR
IntraR	::=	*Intra*(*policy_id*) { Rop *Intra*(*policy_id*)} Eop IntraSet
IntraSet	::=	*Intra*(*policy_id*)$\mid$ $\emptyset$
InterR	::=	*Inter*(*policy_id*) { Rop *Inter*(*policy_id*)} Eop InterSet
InterSet	::=	*Inter*(*policy_id*) $\mid$ *Compl*(*Inter*(*policy_id*))$\mid$ $\emptyset$
SLR	::=	*SL*(*policy_id*) { Rop *SL*(*policy_id*)} Eop SLSet
SLSet	::=	*SL*(*policy_id*) $\mid$ $\emptyset$
Rop	::=	$\cap$ $\mid$ $\cup$ $\mid$ **.**
Eop	::=	$=$ $\mid$ $\neq$

Figure 10.1: Policy Refinement Language BNF

The refinement may add some new policies, but the new policies should not violate the original ones. This means that the security policies after the refinement should be as strict as or stricter than those before the refinement.

In the following subsections, we first present a policy refinement language, which will be used to specify the refinement rules of each pattern. Then, we propose a basic set of refinement patterns for decomposition, aggregation, and elimination of components, connectors, and ports.

10.3.1 Policy Refinement Language

In order to prove that the refinement patterns are correct, we need to provide correct refinement rules for the patterns so that we can verify that the security policy is preserved by applying these patterns. To achieve this, we propose a policy refinement language to formally specify the refinement rules of each pattern. The Backus Normal Form (BNF) of our *Policy Refinement Language (PRL)* is presented in Fig. 10.1. *Intra* and *Inter* are functions that extract the intra-level rules and the inter-level rules of a policy. *SL* extracts the security levels of services on which a security policy has effects. In this work, the security levels TS, S, C, U represent Top Secret, Secret, Confidential, and Unclassified, respectively from high to low. *Compl* returns the complimentary set of rules of a set of rules. For example, the complimentary rule of an inter-level rule, "Downgrading messages from TS to U", is an inter-level rule "Upgrading messages from U to TS". For the relation operator *Rop*, $\cup$ and $\cap$ have semantics of set union and intersection, and "**.**" has the semantics of set concatenation.

Table 10.1: Component Decomposition Patterns for Designing MLS Systems

Decomposition Patterns	**Policy Types of Components**						
	Cm	Cm_1			Cm_2		
		SLS	MSLS	MLS	SLS	MSLS	MLS
Product	SLS	√			√		
	MSLS	√			√		
		√				√	
			√			√	
	MLS	√					√
				√			√
Cascade	SLS	√			√		
				√*			√*
	MSLS		√			√	
				√*			√*
	MLS	√					√
				√	√		
				√			√
Feedback	SLS	√			√		
				√*			√*
	MSLS		√			√	
				√*			√*
	MLS	√					√
				√	√		
				√			√

* Extra inter-level rules added in these cases

10.3.2 Decomposition Patterns

In the refinement process, it is typical to divide an architectural entity into smaller parts through decomposition. Decomposition can be applied to components, connectors, and ports.

Component Decomposition Patterns

A component can be decomposed into two components that are composed through product, cascade, or feedback [184, 277]. These three types of component decomposition are depicted in Fig. 10.2(a-c), respectively. For each component decomposition pattern, depending on the type of security policy (SLS, MSLS, or MLS policy) enforced by the initial component, Cm, there are different combinations of possible types of policies enforced by the subcomponents, Cm_1 and Cm_2, as summarized in Table 10.1. In this section, we discuss the refinement rules of some of the patterns as examples. The refinement rules of all component decomposition patterns are listed in [285]

Product Pattern. The first component decomposition pattern discussed is Product (Fig. 10.2(a)). In this pattern, Cm is constructed by the parallel composition of Cm_1 and Cm_2. Suppose that the policies enforced by components Cm, Cm_1, and Cm_2 are p, p_1, and p_2 correspondingly. Functions In and Out extract the services associated with the input and output ports of a component/connector which enforce policies. The pattern $Product(p, p_1, p_2)$ is defined as:

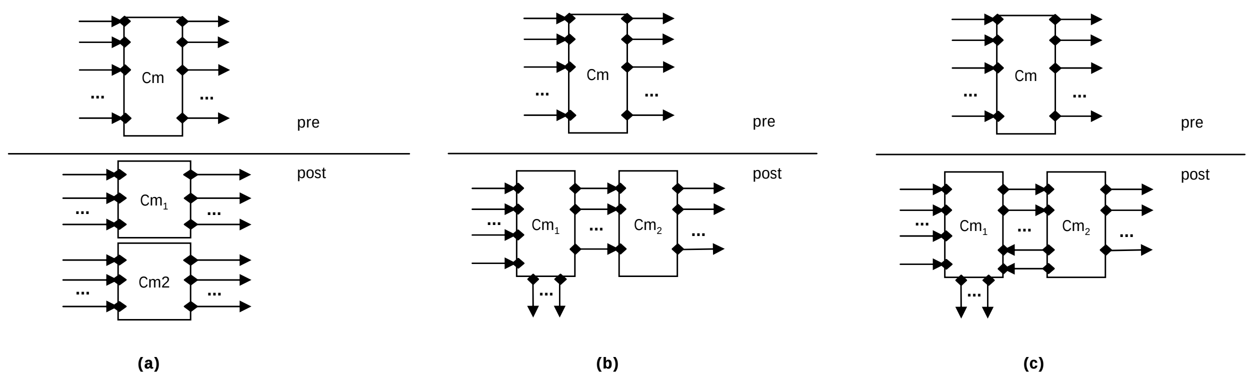

Figure 10.2: Component Decomposition Patterns (a) Product (b) Cascade (c) Feedback

$Product(p, p_1, p_2) \equiv (In(p) = In(p_1) \cup In(p_2)) \wedge (Out(p) = Out(p_1) \cup Out(p_2)) \wedge (In(p_1) \cap In(p_2) = \emptyset) \wedge (Out(p_1) \cap Out(p_2) = \emptyset)$.

For the Product pattern, we discuss one example case in which the component Cm enforces an MSLS policy. In this case, there are three different refinement rules (Product Rule 2-1, 2-2, and 2-3) that can be applied to make the refinement of Cm into two subcomponents, Cm_1 and Cm_2, be correct.

1. **Product Rule 2-1:**$Product(p, p_1, p_2)$, $isMSLS(p)$, $isSLS(p_1)$, $isSLS(p_2)$, $Intra(p_1) = Intra(p)$, $Intra(p_2) = Intra(p)$, $SL(p_1) \cap SL(p_2) = \emptyset$.

   The refinement is correct when Cm_1 and Cm_2 both enforce SLS policies and each component enforces intra-level rules for one different security level, where Cm is an MSLS component processing data at just two different security levels.

2. **Product Rule 2-2:**$Product(p, p_1, p_2)$, $isMSLS(p)$, $isSLS(p_1)$, $isMSLS(p_2)$, $Intra(p_1) = Intra(p)$, $Intra(p_2) = Intra(p)$, $SL(p_1) \cap SL(p_2) = \emptyset$.

   The refinement is correct when one of the components (e.g., Cm_1) enforces an SLS policy while another (e.g., Cm_2) enforces an MSLS policy. In this case, usually Cm is an MSLS component processing data at more than two security levels, Cm_1 enforces intra-level rules for one security level, and Cm_2 enforces intra-level rules for other security levels and the inter-level rule that there is no cross-level information flow.

3. **Product Rule 2-3:**$Product(p, p_1, p_2)$, $isMSLS(p)$, $isMSLS(p_1)$, $isMSLS(p_2)$, $Intra(p_1) \cup Intra(p_2) = Intra(p)$, $Intra(p_1) \cap Intra(p_2) = \emptyset$, $Inter(p_1) = Inter(p)$, $Inter(p_2) = Inter(p)$, $SL(p_1) = SL(p)$, $SL(p_2) = SL(p)$.

   The refinement is correct when both Cm_1 and Cm_2 enforces MSLS policies. In this case, Cm_1 and Cm_2 enforce different part of the intra-level rules of Cm for all security levels of Cm, and enforces the inter-level rule that there is no cross-level information flow. There is another refinement rule which can make the refinement correct. Cm_1 or Cm_2 enforces all intra-level rules in Cm but for only some of the security levels (more than one), and enforces an inter-level rule that there is no cross-level information flow. The pattern with this refinement rule is not a basic pattern, which will be discussed in Section 10.4.

In all of the Product decomposition cases, we decompose the original component into two subcomponents. The decomposition can separate the security concerns by separating the processing of difference security levels. When this happens, the result is a more secure system. However, sometimes the decomposition does not separate security levels. In that case we still need to enforce inter-level rules as discussed above. In either case, the security policies enforced by the subcomponents, when composed, can not violate that of the main component.

Cascade Pattern. The second decomposition pattern is Cascade (Fig. 10.2(b)). In this pattern, Cm is constructed through serial composition of Cm_1 and Cm_2. Suppose that the policies enforced by component Cm, Cm_1, connector between Cm_1 and Cm_2, and component Cm_2 are correspondingly p, p_1, p_2, and p_3. Policy p_2 is application independent. The pattern $Cascade(p, p_1, p_2, p_3)$ is defined as: $Cascade(p, p_1, p_2, p_3) \equiv (In(p) = In(p_1) \cup In(p_3) - Out(p_2)) \wedge (Out(p) = Out(p_1) \cup Out(p_3) - In(p_2)) \wedge (In(p_1) \cap In(p_3) = \emptyset) \wedge (Out(p_1) \cap Out(p_3) = \emptyset)$.

For the Cascade pattern, we discuss one example case in which the component Cm enforces an SLS policy. In this case, there are two different refinement rules (Cascade Rule 1-1 and 1-2) that can be applied to make the refinement be correct.

1. **Cascade Rule 1-1:**$Cascade(p, p_1, p_2, p_3)$, $isSLS(p)$, $isSLS(p_1)$, $isSLS(p_2)$, $isSLS(p_3)$, $Intra(p_1) \centerdot Intra(p_2) \centerdot Intra(p_3) = Intra(p)$, $SL(p_1) = SL(p)$, $SL(p_3) = SL(p)$.

   The refinement is correct when Cm_1, Cm_2, and the connector between them all enforce SLS policies and they together enforce the intra-level rules of Cm.

2. **Cascade Rule 1-2:**$Cascade(p, p_1, p_2, p_3)$, $isSLS(p)$, $isMLS(p_1)$, $isSLS(p_2)$, $isMLS(p_3)$, $Intra(p_1) \centerdot Intra(p_2) \centerdot Intra(p_3) = Intra(p)$, $Inter(p_1) = Compl(Inter(p_3))$.

   The refinement is correct when both Cm_1 and Cm_2 enforce MLS policies which are complimentary to each other, and the original intra-level rules are enforced by Cm_1, Cm_2, and the connector between them. In this case, both Cm_1 and Cm_2 need to enforce additional inter-level rule specific and maintain the original SLS policy of Cm. Such a refinement could occur if one of these two MLS components enforces an inter-level rule to down-grade while the other enforces a complimentary inter-level rule to up-grade. A typical example of applying this refinement rule of the Cascade pattern is shown in Fig. 10.3. In this example, Cm is an abstract SLS communication component that takes in messages at TS level and transfers the messages to output them also at TS. A possible correct refinement of Cm would involve encrypted communication over an insecure channel (connector Cn). We could have Cm_1 be an encryption device which takes in TS messages, encrypts them with a key for TS messages, then outputs U messages, and Cm_2 be a decryption device which takes in U messages, decrypts them with TS keys, then outputs TS messages. Obviously, Cm_1 and Cm_2 are both components that enforce MLS policies and with their composition, the transfer of TS messages is secure, with no message leakage to lower levels.

Feedback Pattern. The final decomposition pattern is Feedback (Fig. 10.2(c)). In this pattern, Cm is constructed by two communication components Cm_1 and Cm_2. Suppose that the policies enforced by component Cm, Cm_1, connector between Cm_1 and Cm_2, component Cm_2, connector between Cm_2 and Cm_1 are correspondingly p, p_1, p_2, p_3, and p_4. Policies p_2 and p_4 are application independent. The pattern $Feedback(p, p_1, p_2, p_3, p_4)$ is defined as: $Feedback(p, p_1, p_2, p_3, p_4) \equiv (In(p) = In(p_1) \cup In(p_3) - Out(p_2) - Out(p_4)) \wedge (Out(p) = Out(p_1) \cup Out(p_3) - In(p_2) - In(p_4)) \wedge (In(p_1) \cap In(p_3) = \emptyset) \wedge (Out(p_1) \cap Out(p_3) = \emptyset)$.

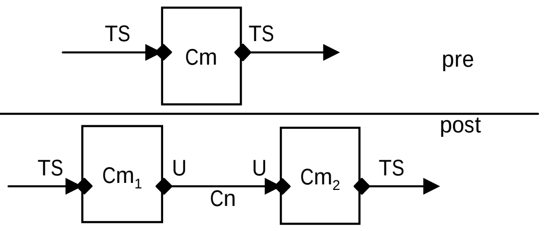

Figure 10.3: Component Decomposition Cascade Pattern Example

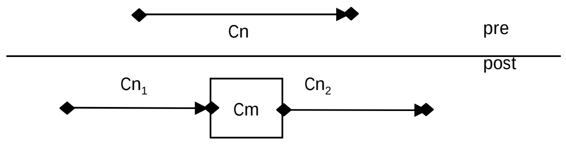

Figure 10.4: Connector Decomposition ConnDec Pattern

The correct refinement cases summarized in Table 10.1 are similar to those for Cascade pattern and are omitted here to save space. However, it is important to note that some security policies are not preserved under feedback composition [184, 277, 183]. We discussed feedback composition in our previous work [288].

Connector and Port Decomposition Patterns

As mentioned in Section 10.2, in the final concrete architecture, a connector does not perform any application specific computations. Therefore, during the architectural refinement, for any connector that enforces application specific security policy, we should refine this connector using ConnDec pattern (Fig. 10.4). In pattern ConnDec, connector Cn is decomposed into connectors Cn_1 and Cn_2 and component Cm. After the ConnDec refinement, all application specific security policies enforced by Cn are enforced by component Cm and no application specific security policy is enforced by connectors Cn_1 and Cn_2. Suppose that the policies enforced by Cn, Cn_1, Cm, and Cn_2 are correspondingly p, p_1, p_2, and p_3. Policies p_1 and p_3 are application independent. The pattern $ConnDec(p, p_1, p_2, p_3)$ is defined as: $ConnDec(p, p_1, p_2, p_3) \equiv (In(p) = In(p_1)) \wedge (Out(p) = Out(p_3)) \wedge (In(p_2) = Out(p_1)) \wedge (In(p_3) = Out(p_2))$.

For the ConnDec pattern, there is one refinement rule to make the refinement correct.

ConnDec Rule: $ConnDec(p, p_1, p_2, p_3)$, $Intra(p_1) \centerdot Intra(p_2) \centerdot Intra(p_3) = Intra(p)$, $Inter(p_1) \centerdot Inter(p_2) \centerdot Inter(p_3) = Intra(p)$.

The refinement is correct when the connectors Cn_1 and Cn_2 and the component Cm together enforce the intra-level rules and inter-level rules of Cn, and Cm must enforce the application specific rules.

When a connector enforces application independent security policy, it is correct to refine this connector into a serial composition of two connectors if the two connectors together enforce the application independent security policies (not pictured here).

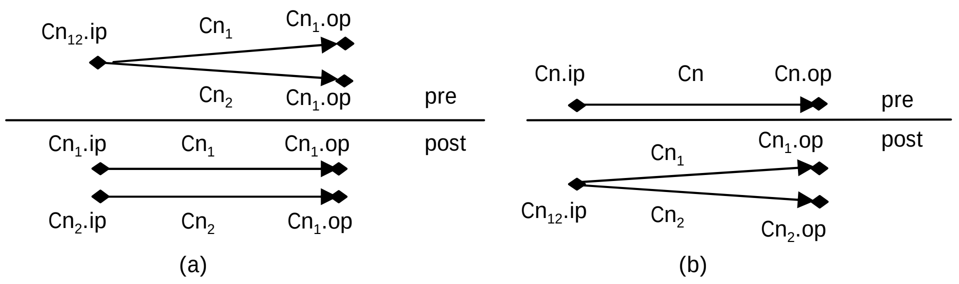

Figure 10.5: Port Decomposition Patterns

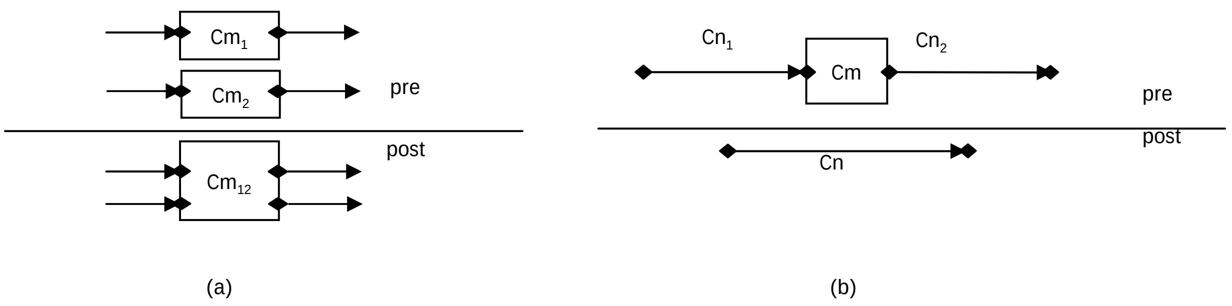

Figure 10.6: Component/Connector Aggregation Patterns (a) ComAgg (b) ComConnAgg

For port decomposition, we decompose a port into multiple ports, splitting the services of the original port to associate them with the new ports. When a port of a component decomposes, nothing changes in the component and no security policy will be violated. For the connector to this port, in one case, such a port decomposition causes the connector to decompose (Fig. 10.5 (b)). In this case, we need to refine the security policies enforced by the connector Cn to ensure they are enforced by Cn_1 and Cn_2. Possible inter-level rules enforced by Cn_1 and Cn_2 should not violate those enforced by Cn.

In another case, a port decomposition does not cause the connector to decompose (Fig. 10.5(a)). In this case, connectors Cn_1 and Cn_2 remain the same, meaning the security policies enforced by these two connectors remain the same.

10.3.3 Aggregation Patterns

During the process of architectural refinement, we may find it is necessary to merge two more abstract entities into a single lower level entity. If two components are not connected directly, when we merge them as one component, we say we aggregate them. If two components are connected directly, we can consider them as part of a bigger component, but not an aggregation. Aggregation can be applied to components, connectors, and ports.

ComAgg Pattern. The component aggregation pattern we provide in this paper is ComAgg (Fig. 10.6(a)). Suppose that the policies enforced by components Cm_1, Cm_2, and Cm_{12} are p_1, p_2, and p correspondingly. The pattern $ComAgg(p_1, p_2, p)$ is defined as: $ComAgg(p_1, p_2, p) \equiv (In(p) = In(p_1) \cup In(p_2)) \wedge (Out(p) = Out(p_1) \cup Out(p_2))$.

Table 10.2: Component Aggregation Patterns for Designing MLS Systems

Policy Type		**Policy Type**		
Cm_1	Cm_2	Cm_{12}		
		SLS	MSLS	MLS
SLS	SLS	$\surd$		
			$\surd^*$	
SLS	MSLS		$\surd^*$	
SLS	MLS			$\surd^*$
MSLS	MLS			$\surd$
MLS	MLS			$\surd$

* Extra inter-level rules added in these cases

For the ComAgg pattern, depending on the types of security policy enforced by the initial component, Cm_1 and Cm_2, the possible type of policy enforced by the new component, Cm_{12}, is different, as summarized in Table 10.2. In this section, we discuss the refinement rules of some example cases of the ComAgg pattern, and the refinement rules of all cases are listed in [**?**].

In the example case, either Cm_1 or Cm_2 is an MSLS component while the other is an MLS component. In this case, there are two different refinement rules (ComAgg Rule 4-1 and 4-2) that can be applied to make the aggregation of two components into a new one be correct.

1. **ComAgg Rule 4-1:**$ComAgg(p_1,p_2,p)$, $isMSLS(p_1)$, $isMLS(p_2)$, $isMLS(p)$, $Intra(p_1) \cup Intra(p_2) = Intra(p)$, $Inter(p_1) \cap Inter(p_2) = Inter(p)$, $SL(p_1) \cap SL(p_2) \neq \emptyset$.

   Cm_{12} should be an MLS component that enforces all intra-level rules of Cm_1 and Cm_2, and the MLS inter-level rules of Cm_{12} should not violate those of Cm_1 and Cm_2. When there is a conflict between the MLS inter-level rule and the MSLS inter-level rule of the original components (e.g., according to the MLS inter-level rule, there is cross-level information flow between TS and S, but according to the MSLS inter-level rule, there should be no cross-level information flow), we require the stricter rule be enforced. This means that Cm_{12} must enforce the MSLS inter-level rule instead of the MLS inter-level rule when conflict occurs.

2. **ComAgg Rule 4-2:**$ComAgg(p_1,p_2,p)$, $isMSLS(p_1)$, $isMLS(p_2)$, $isMLS(p)$, $Intra(p_1) \cup Intra(p_2) = Intra(p)$, $Inter(p_1) \cup Inter(p_2) = Inter(p)$, $SL(p_1) \cap SL(p_2) = \emptyset$.

   Cm_{12} should be an MLS component that enforces all intra-level rules of Cm_1 and Cm_2. And since there is no conflict between the MLS rule and the MSLS rule of the original two components, Cm_{12} should enforces all the MLS and MSLS inter-level rules of both original components.

It is important to note that in ComAgg, though the two original components do not interact directly with each other, they both interact with their environment, and in some cases, their aggregation may cause a subtle situation that needs to be examined closely, as we discussed in greater detail in our previous work [288].

ComConnAgg Pattern. In addition to ComAgg pattern, we propose a ComConnAgg pattern (Fig. 10.6(b)) to aggregate component and connectors. In this pattern, connectors Cn_1 and Cn_2 and component Cm are aggregated to a new connector Cn. Suppose that the policies enforced by Cn,

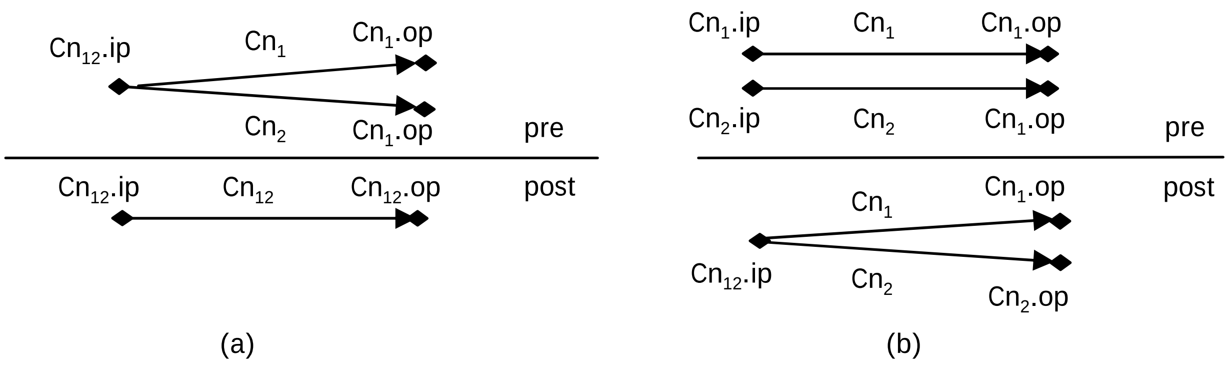

Figure 10.7: Port Aggregation Patterns

Cn_1, Cm, and Cn_2 are p, p_1, p_2, and p_3 correspondingly. This pattern can be applied only when all these policies are application independent. The pattern is defined as: $ComConnAgg(p_1, p_2, p_3, p)$ $\equiv (In(p) = In(p_1)) \wedge (Out(p) = Out(p_3))$. This aggregation is correct if and only if Cn is trusted to enforce all the security policies of Cn_1, Cn_2, and Cm, as specified by the ComConnAgg Rule:

ComConnAgg Rule: $ComConnAgg(p_1, p_2, p_3, p)$,
$Intra(p_1) \centerdot Intra(p_2) \centerdot Intra(p_3) = Intra(p)$,
$Inter(p_1) \centerdot Inter(p_2) \centerdot Inter(p_3) = Inter(p)$.

Also, two connectors composed serially can be aggregated to one connector if the new connector does not violate any security policies of the two original connectors.

Port Aggregation Patterns. For port aggregation, we merge the services of multiple ports to associate them with a new port. When the ports of a component are aggregated, nothing changes in the component and the security policy is not violated. For the connectors to these ports, in one case, such a port aggregation causes the connectors to aggregate as well (Fig. 10.7 (a)). In this case, Cn_{12} should enforce all the security policies of Cn_1 and Cn_2. Only when the two connectors Cn_1 and Cn_2 are processing data at the same security level will no inter-level rule be enforced by Cn_{12}; otherwise, there should always be inter-level rules enforced by Cn_{12}, and they should not violate the inter-level rules of Cn_1 and Cn_2.

In another case, ports aggregation does not cause connectors with these ports to aggregate (Fig. 10.7 (b)). In this case, after the port aggregation, connectors Cn_1 and Cn_2 remain the same, meaning all security policies of these two connectors remain the same.

10.3.4 Elimination Patterns

Simply stated, for a connector with no semantics, meaning no functionality is provided by it, we can eliminate this connector without violating any security policy. Elimination of a component is correct if the component has no connection to other components. Also, elimination of a port is correct if it has no connection to its environment.

Table 10.3: Example $Level_1$ Patterns

Patterns	**Policy Types**			$Level_0$ **Rules**
Product	MSLS (p)	MSLS (p_1)	MSLS (p_2)	*Product Rule* 2-2 *ComAgg Rule* 1-2
Product	MLS (p)	MSLS (p_1)	MLS (p_2)	*Product Rule* 3-1 *ComAgg Rule* 1-2
Cascade	MSLS (p)	SLS (p_1)	MSLS (p_3)	*Product Rule* 2-1/2-2 *Cascade Rule* 1-1 *ComAgg Rule* 1-2/2
Cascade	MLS (p)	MSLS (p_1)	MLS (p_3)	*Product Rule* 3-2 *Cascade Rule* 3-1 *ComAgg Rule* 1-2 *ComAgg Rule* 5
Feedback	MLS (p)	MSLS (p_1)	MLS (p_3)	*Product Rule* 3-2 *Feedback Rule* 3-1 *ComAgg Rule* 1-2 *ComAgg Rule* 5
ComAgg	MSLS (p)	MSLS (p_1)	MSLS (p_2)	*Product Rule* 2 *ComAgg Rule* 2

10.4 Hierarchy of Refinement Patterns

In this section, we propose the hierarchy of the refinement patterns. In our work, the $Level_0$ pattern set consists of all the patterns presented in the previous section. It is a set of refinement patterns on which other refinement patterns can be built. Intuitively, the architects can build their own patterns in any $Level_{i+1}$ set based on the patterns from $Level_0$ set to $Level_i$ set. In the policy-based architectural refinement verification system that we will develop as a future work, we will first prove that the refinement rules of the patterns in $Level_0$ are correct and provide these rules as the axioms of the verification system. The refinement rules of patterns in $Level_1$ and above are theorems that need to be proved based on the axioms provided. In this way, the verification of patterns in $Level_{i+1}$ can reuse the verification of patterns in lower level sets, and verification efforts can be reduced.

Some examples of $Level_1$ patterns, but not exhaustive, are presented in Table 10.3, and their refinement rules are listed in [285]. For each example $Level_1$ pattern, the $Level_0$ Rules column in Table 10.3 indicates the $Level_0$ rules that can be applied to verify its refinement rule. We discuss one example $Level_1$ pattern, *Cascade* ($MSLS \rightarrow SLS + MSLS$) to informally justify that the refinement rule of this pattern can be verified based on the rules of $Level_0$ patterns. We leave the formal verification of the listed $Level_1$ patterns as our future work.

The *Cascade* ($MSLS \rightarrow SLS + MSLS$) pattern says that when Cm is an MSLS component, either Cm_1 or Cm_2 could be an MSLS component while the other is an SLS component (Fig. 10.8). This pattern is not a $Level_0$ pattern, but a $Level_1$ pattern since such a decomposition pattern can be achieved by applying a set of refinement patterns sequentially as depicted in Fig. 10.9.

The rule $Cascade{-}L_1\ Rule1(MSLS) : Cascade(p, p_1, p_2, p_3), isMSLS(p), isSLS(p_1), isMSLS(p_3),$ $Intra(p_1) \centerdot Intra(p_2) \centerdot Intra(p_3) = Intra(p),$

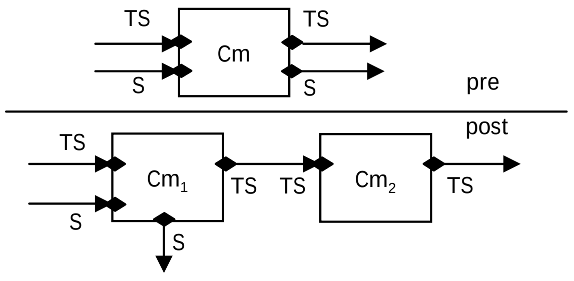

Figure 10.8: Cascade $Level_1$ Pattern (MSLS→SLS+MSLS)

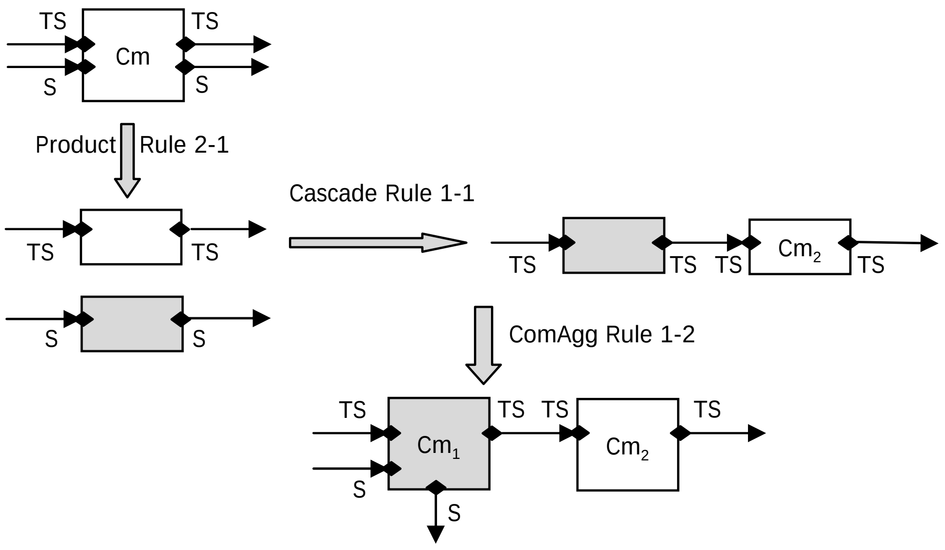

Figure 10.9: Achieved through Applying a Set of $Level_0$ Refinements

$Inter(p_3) = Inter(p)$ can be verified through applying a set of $Level_0$ rules, Product Rule 2-1, Cascade Rule 1-1, and ComAgg Rule 1-2, sequentially.

10.5 A MILS Application Design Example

In this section, we present the process of designing a MILS application system to illustrate how to design an MLS system by applying the refinement patterns described in the previous sections. The system is simple enough to avoid complexity for the purposes of discussion but sufficient to illustrate the concepts of our approach. The security policies of this example are specified informally in written text and we justify each step of the refinement by informally verifying that the refinement rule of the applied pattern is satisfied. In the future work, we will use a policy specification language to specify policies so that it can be formally verified that the refinement rules of each step are satisfied. The specifications of the system are: A distributed MILS application system that allows users with different security levels (TS and S) to store and retrieve data on an MLS database using legal CORBA messages (i.e., legal means read, write methods for users and reply method for database). Assume user U_1 is at TS level while U_2 is at S level and the database DB is remotely accessible by both users. The security policy is: 1. inter-level rule: store/retrieve data obeying

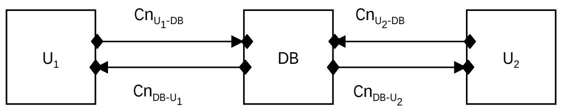

Figure 10.10: Abstract Architecture of the Application Example

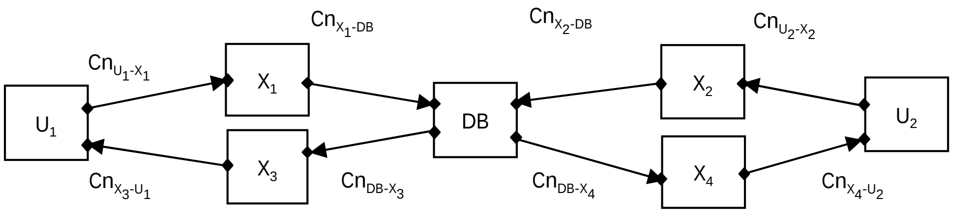

Figure 10.11: Architecture of the Example System after Step 1

BLP model; 2. intra-level rule: all messages sent should be legal CORBA messages.

The abstract architecture of this system is depicted in Fig. 10.10. In this abstract architecture, connector Cn_{U_1-DB}, Cn_{U_2-DB}, Cn_{DB-U_1}, and Cn_{DB-U_2} enforce security policies listed in Table 10.4, No. 1, 2, 3, and 4 respectively.

Following are several refinement steps in the process of the application system design which illustrate how to obtain a correct concrete system architecture that preserves the original security policies.

Step 1. In the abstract architecture, connectors Cn_{U_1-DB}, Cn_{U_2-DB}, Cn_{DB-U_1}, and Cn_{DB-U_2} enforce application specific policies, we apply the connector decomposition pattern ConnDec to each connector to put the application specific policies onto components. After the refinement, the new architecture is depicted in Fig. 10.11 and the policies enforced by the added components are listed in Table 10.4, No. 5 to 8. The new connectors between components do not enforce security policy. The new components can be considered as guards for application specific communications between users and DB. It is obvious that the *ConnDec Rule* is satisfied by the policies before and after *ConnDec* pattern is applied to each original connector.

Step 2. We apply component aggregation pattern ComAgg to merge two guards X_1 and X_3 into a single bi-directional guard as we did with the other two guards, X_2 and X_4. In the new architecture (Fig. 10.12), the policy enforced by component X_{13}, No. 9, do not violate the policies enforced by X_1 and X_3, therefore, the *ComAgg Rule* 3 is satisfied. Similarly, the policy enforced by component X_{24}, No.10, does not violate the policies enforced by X_2 and X_4, and the *ComAgg Rule* 3 is also satisfied.

Step 3. We further apply ComAgg pattern to merge X_{13} and X_{24} to a component X which processes data at different security levels, and the new architecture is depicted in Fig. 10.13(a). It is straightforward that the security policies enforced by X_{13} and X_{24} (before refinement) and the security policy enforced by X (No. 11, after refinement) satisfy the refinement rule *ComAgg Rule* 5-2. Therefore, this step of refinement is correct.

Step 4. Since DB is not at the same location as U_1 and U_2, we apply component decomposition

Table 10.4: Application Policies Enforcement Table

Com/ Conn	**No.**	**Policy**	
		Intra-level Rules	**Inter-level Rules**
Cn_{U_1-DB}	1	Message is legal CORBA message.	U_1 read and write DB according to BLP.
Cn_{U_2-DB}	2	Message is legal CORBA message.	U_2 read and write DB according to BLP.
Cn_{DB-U_1}	3	Message is legal CORBA message.	
Cn_{DB-U_2}	4	Message is legal CORBA message.	
X_1	5	Message is legal CORBA message.	U_1 read/write TS data and read S data.
X_2	6	Message is legal CORBA message.	U_2 read/write S data and write TS data.
X_3	7	Message is legal CORBA message.	
X_4	8	Message is legal CORBA message.	
X_{13}	9	Message between U_1 and DB is legal CORBA message.	U_1 read/write TS data and read S data.
X_{24}	10	Message between U_2 and DB is legal CORBA message.	U_2 read/write S data and write TS data.
X	11	Message between U_1/U_2 and DB is legal CORBA message.	U_1 read/write TS data and read S data; U_2 read/write S data and write TS data.
X'	12	Message between U_1/U_2 and DB is legal CORBA message.	U_1 read/write TS data and read S data; U_2 read/write S data and write TS data.
	13		Downgrade TS/S message to U by encrypting with TS/S key; upgrade U message to TS/S by decrypting with TS/S key.
X''	14	Message between U_1/U_2 and DB is legal CORBA message.	U_1 read/write TS data and read S data; U_2 read/write S data and write TS data.
$TNIU$	15		Downgrade TS/S message to U by encrypting with TS/S key; upgrade U message to TS/S by decrypting with TS/S key.
MMR	16	Forward different types of messages to the correct type checking components.	Messages from different partitions must be labelled correctly with security levels of the partitions.
GG	17	Message received must be legal CORBA message.	U_1 read/write TS data and read S data; U_2 read/write S data and write TS data.
DB	18	Reply with legal CORBA message.	Reply to requests with data at proper classification levels.

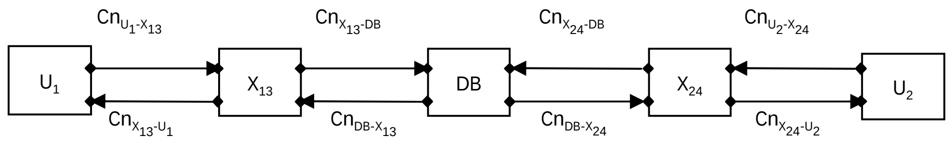

Figure 10.12: Architecture of the Example System after Step 2

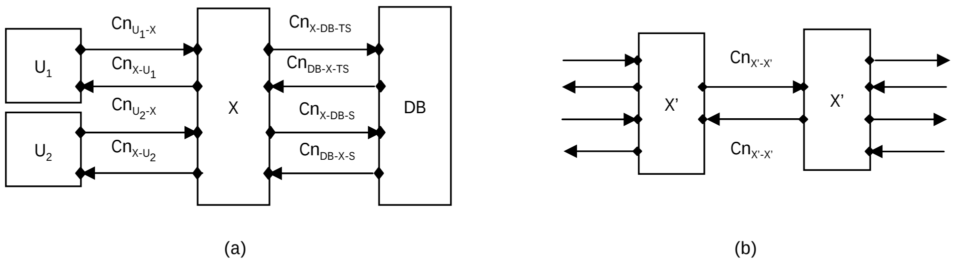

Figure 10.13: Architecture of the Example System after (a) Step 3 (b) Step 4

Feedback pattern to decompose X by putting two same component X' at two different locations. Additional inter-level rules need to be enforced by X' to down-grade/up-grade. As long as the assumption that the encrypted messages are not leaked by the insecure network, the combination of the X', an insecure network, and X' will provide a secure communication, as shown in Fig. 10.13(b). The refinement rule *Feedback Rule* 3-2 is satisfied by the policies before and after the refinement.

Step 5. We continue the refinement by applying the component decomposition pattern Feedback to decompose X' into a composition of X'' and $TNIU$ (Trusted Network Interface Unit) to separate guarding from secure communication, as shown in Fig. 10.14(a). The component $TNIU$ enforces the MLS policy, No. 15, to downgrade/upgrade all messages it receives before forwarding them so that it can provide secure communications between different locations. Component X'' enforces the security policy No. 14, which performs the secure guarding and does not violate the security policies No. 12 and 13 of X'. Connectors $Cn_{X''-TNIU}$ and $Cn_{TNIU-X''}$ can transfer messages using direct process calls. The *Feedback Rule* 3-2 is satisfied during the refinement.

Step 6. We apply decomposition pattern Feedback to decompose X'' into two other generic se-

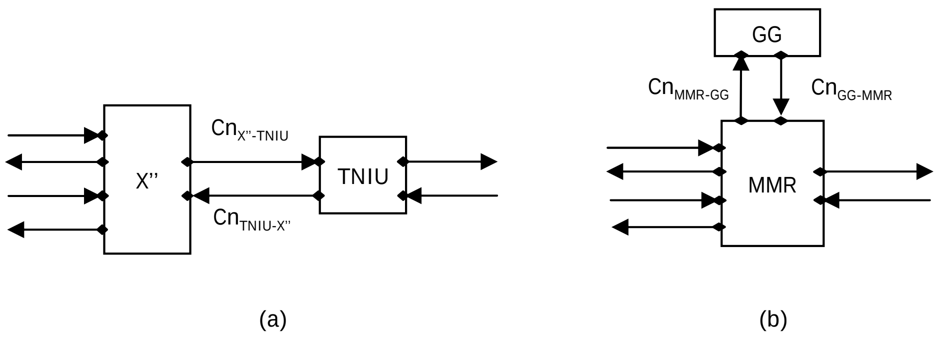

Figure 10.14: Part of Architecture of the Example System after (a) Step 5 (b) Step 6

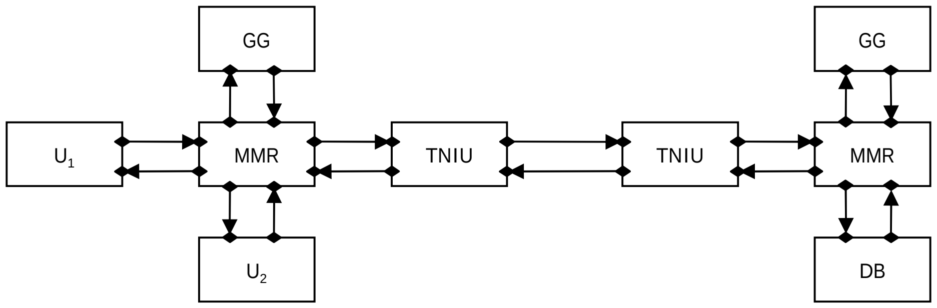

Figure 10.15: Final Concrete Architecture of the Example System

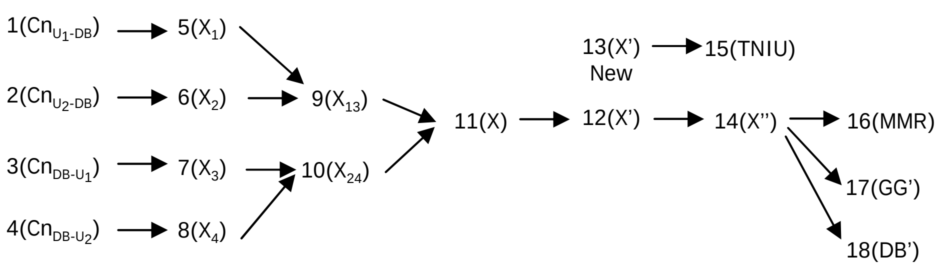

Figure 10.16: Trace of Security Policies Refinement and Enforcement

cure building blocks, MMR and GG, depicted in Fig. 10.14(b). In our work, MMR (MILS Message Router) does the component identification, message labeling and routing of typed messages. The GG (GIOP Guard, a guard for checking legal CORBA messages) does CORBA message checking. Specifically, MMR is an MSLS component that enforces the security policy No. 16, and GG is an MLS component enforces security policies No. 17. Connectors Cn_{MMR-GG} and Cn_{GG-MMR} can transfer messages using direct process calls. In this step of refinement, $Feedback - L_1\ Rule1$ is satisfied.

Final. After all these steps, we take one additional refinement step by applying port aggregation patterns to aggregate the ports and connectors between the MMR and DB to achieve the final concrete architecture shown in Fig. 10.15. The security levels associated with the ports of the connectors between MMR and DB are TS and S.

Through informally justifying that the refinement rule of the applied refinement pattern is satisfied for each step of refinement, we can informally guarantee that this concrete architecture is a correct design of the application example, if components MMR, GG, $TNIU$, and DB are designed to enforce the security policies in Table 10.4.

The trace of how the original security policies are refined and enforced during the design process is presented in Fig. 10.16. The numbers in the figure represent the respective security policies in Table 10.4.

By walking through the design of this application example, we can see that our refinement patterns provide guidance for the design of MLS systems by being applied to each step of architectural refinement, and the refinement rule of each applied pattern can be a criterion to justify that the refinement is correct.

10.6 Conclusions and Future Work

In this paper, we extend our previous work of a set of correct architectural refinement patterns for the design of MLS systems. We present a policy refinement language, PRL, to specify the refinement rules of patterns, and propose the hierarchy of the patterns. The rules for the patterns in $level_0$ are the axioms in the policy-based architectural refinement verification system, while the rules for the patterns in higher levels are theorems that can be proved based on the axioms and that can be reused by architects to verify the correctness of the refinement of application policies.

Our current work only provides an informal framework of policy-based architectural refinement technique for the design of MLS system. As future work, we will work on the formal verification of our approach. We will provide the formal definition of correct security policy refinement for the design of MLS system, and provide the formal semantics of PRL. Based on these, the refinement rules of each refinement pattern will be proved to be correct, as well as the pattern hierarchy. Then, we will extend an existing policy specification language to specify MLS system application policies, so that when we apply our refinement pattern to each step of the design of an example system, we can formally verify that the refinement rule of the pattern is satisfied, which means that the policies are refined correctly and enforced during the MLS systems design.

Part III

Bibliography of cited work and Abstracts of Publications

Bibliography

[1] Mathematical markup language (mathml) 2.0. Recommendation REC-xml-20040204.

[2] Functional safety of electrical/ electronic/ programmable electronic safety-related systems. Technical Report IEC 61508, International Electrotechnical Commission, Jan. 1998. Parts 1 to 7.

[3] Xsl transformations (XSLT). Recommendation REC-xslt-19991116, Nov. 1999.

[4] Final annual report for clarification of DO-178B. Technical Report DO-248B, Dec. 2001.

[5] Xml metadata interchange specification (XMI). Technical Report 2.1, Mar. 2001.

[6] Extensible markup language (xml) 1.0. Recommendation REC-xml-20040204, W3C, Feb. 2004.

[7] CHISEL, Mar. 2006.

[8] EUROCAE website, Mar. 2006.

[9] FrontEndArt, 2006.

[10] Graph eXchange Language, Feb. 2006.

[11] The Graphviz: Graph visualization software, Mar. 2006.

[12] KAON2, 2006.

[13] Project Bauhaus, 2006.

[14] RTCA website, Mar. 2006.

[15] RuleML, 2006.

[16] Software Engineering Research Laboratory, 2006.

[17] Visual Prolog, Feb. 2006.

[18] W3C, 2006.

[19] ISO/IEC 10181-4. Information-technology - Open systems interconnection - Security frameworks in open systems-Security frameworks in open systems - Part 4: Non-repudiation. Technical report, ISO/IEC, 1996.

[20] ISO/IEC DIS 13888-1. Information-technology - Security techniques - Non-repudiation - Part 1: General model. Technical report, ISO/IEC JTC1/SC27 N1503, November 1996.

[21] ISO/IEC DIS 13888-3. Information-technology - Security techniques - Non-repudiation - Part 2: Using asymmetric techniques. Technical report, ISO/IEC JTC1/SC27 N1505, November 1996.

[22] ISO/IEC 5th CD 13888-2. Information-technology - Security techniques - Non-repudiation - Part 3: Using asymmetric techniques. Technical report, ISO/IEC JTC1/SC27 N1505, November 1996.

[23] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 122–136, 1994.

[24] A. Al-Muhaitheef. *The Firewall Mobile Customs Agents: A Distributed Firewall Architecture.* PhD thesis, Dept. of Computer Science, University of Idaho, Aug. 2002.

[25] J. Alves-Foss. *Mechanical Verification of Secure Distributed System Specifications.* PhD thesis, Department of Computer Science, University of California, Davis, 1991.

[26] J. Alves-Foss. The architecture of secure systems. In *Hawa'ii International Conference on System Sciences*, pages 307–316, Jan. 1998.

[27] J. Alves-Foss. Multi-Protocol Attacks and the Public Key Infrastructure. In *Proc. National Information Systems Security Conference*, pages 566–576, October 1998.

[28] J. Alves-Foss. Cryptographic protocol engineering: Building security from the ground up. In *Proc. International Conference on Internet Computing 2000*, pages 371–377, Jun. 2000.

[29] J. Alves-Foss, D. Conte de Leon, and P. Oman. Experiments in the use of XML to enhance traceability between object-oriented design specs. and source code. In *Proc. Of the 35th Hawaii Intl. Conf. On System Sciences*, pages 3959 – 3966, 2002.

[30] J. Alves-Foss, W.S. Harrison, P. Oman, and C. Taylor. The MILS architecture for high assurance embedded systems. *International Journal of Embedded Systems*, 2(3/4):239–247, 2006.

[31] J. Alves-Foss and K. Levitt. Verification of secure distributed systems in higher order logic: A modular approach using generic components. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 122–135, 1991.

[32] J. Alves-Foss and C. Taylor. An analysis of the GWV security policy. In *ACL2 Workshop 2004*, 2004.

[33] Jim Alves-Foss, W. Scott Harrison, Paul Oman, and Carol Taylor. The MILS architecture for high-assurance embedded systems. *International Journal of Embedded Systems*, in press, 2007.

[34] Jim Alves-Foss, Carol Taylor, and Paul W. Oman. A multi-layered approach to security in high assurance systems. In *Proc. Hawaii Systems Sciences Conference*, 2004.

[35] Vincenzo Ambriola and Alina Kmiecik. Architectural transformations. In *SEKE '02: Proceedings of the 14th international conference on software engineering and knowledge engineering*, pages 275–278, New York, NY, USA, 2002. ACM Press.

[36] Ben Ames. Real-time software goes modular. *Military & Aerospace Electronics*, 14(9), Sept. 2003. URL: http://www.ghs.com/download/articles/GHS_RTOS_modular_090103.pdf.

[37] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik. On the performance of group key agreement protocols. In *Proc. 22nd IEEE Conference on Distributed Computing Systems*, Jul. 2002.

[38] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik. On the performance of group key agreement protocols. *ACM Transactions on Information and System Security*, 7(3):457–488, 2004.

[39] J. P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, USAF Electronic Systems Div., Bedford, Mass., Oct. 1972.

[40] James P. Anderson. Computer security technology planning study. Technical report, Fort Washing, PA, 1972.

[41] Kenneth M. Anderson, Susanne A. Sherba, and William Van Lepthien. Towards large-scale information integration. In *Proc. 24th Int'l Conf. on Software Engineering (ICSE'02)*, pages 524–534, May 2002.

[42] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, and Andrea De Lucia. Information retrieval models for recovering traceability links between code and documentation. In *Proc. 16th Int'l Conf. on Software Maintenance (ICSM'00)*, page 40, Oct. 2000.

[43] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, and Andrea De Lucia. Maintaining traceability links during object-oriented software evolution. *Software Practice and Experience*, 31(4):331–355, Apr. 2001.

[44] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, Oct. 2002.

[45] Giuliano Antoniol, Gerardo Canfora, and Andrea De Lucia. Maintaining traceability during object-oriented software evolution: a case study. In *Proc. 15th Int'l Conf. on Software Maintenance (ICSM'99)*, pages 211–219, Aug. 1999.

[46] Giuliano Antoniol, Gerardo Canfora, Andrea De Lucia, Gerardo Casazza, and Ettore Merlo. Tracing object-oriented code into functional requirements. In *Proc. 8th Int'l Workshop on Program Comprehension (IWPC'00)*, pages 79–86, Jun. 2000.

[47] Giuliano Antoniol, Gerardo Canfora, Andrea De Lucia, and Ettore Merlo. Recovering code to documentation links in object-oriented systems. In *Proc. 6th Working Conf. on Reverse Engineering (WCRE'99)*, pages 136–144, Oct. 1999.

[48] Giuliano Antoniol, Bruno Caprile, Alessandra Potrich, and Paolo Tonella. Design-code traceability for object-oriented systems. *Annals of Software Eng.*, 9(1-2):35–58, Mar. 2000.

[49] Giuliano Antoniol, Bruno Caprile, Alessandra Potrich, and Paolo Tonella. Design-code traceability recovery: Selecting the basic linkage properties. *Science of Comput. Program.*, 40(2-3):213–234, Jul. 2001.

[50] Giuliano Antoniol, Gerardo Casazza, and Aniello Cimitile. Traceability recovery by modeling programmer behavior. In *Proc. 7th Working Conf. on Reverse Engineering (WCRE'00)*, pages 240–247, Nov. 2000.

[51] Giuliano Antoniol, Alessandra Potrich, Paolo Tonella, and Roberto Fiutem. Evolving object-oriented design to improve code traceability. In *Proc. 7th Int'l Workshop on Program Comprehension (IWPC'99)*, pages 151–160, May 1999.

[52] ARINC. *ARINC Specificaiton 653: Avionics Application Software Standard Interface.* Aeronautical Radio, Inc., Jan. 1997.

[53] Common Criteria Recognition Arrangement. *Common Criteria for Information Technology Security Evaluation Version 2.1*, 1999.

[54] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. Authenticated group key agreement and friends. In *5th ACM Conference on Computer and Communications Security*, pages 17–26, San Francisco, CA, Nov. 1998. ACM Press.

[55] T. Aura. Strategies against replay attacks. In *Proceedings of the 10th IEEE Computer Society Foundations Workshop*, pages 59 – 68, Rockport, MA, June 1997. IEEE Computer Society Press.

[56] Algirdas Avižienis, Jean Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable and Secure Comput.*, 1(1):11–33, 2004.

[57] Alguirdas Avižienis, Jean Claude Laprie, and Brian Randell. Fundamental concepts of computer systems dependability. In *Proc. Workshop on Robot Dependability*, Seoul, Korea (South), May 2001.

[58] Greg J. Badros. JavaML: a markup language for Java source code. *Computer Networks*, 33(1–6):159–177, 2000.

[59] Somo Banerjee, Chris A. Mattmann, Nenad Medvidovic, and Leana Golubchik. Leveraging architectural models to inject trust into software systems. In *SESS '05: Proceedings of the 2005 workshop on software engineering for secure systems building trustworthy applications*, pages 1–7, New York, NY, USA, 2005. ACM Press.

[60] K. Suzanne Barber, Tom Graser, and Jim Holt. Enabling iterative software architecture derivation using early non-functional property evaluation. In *ASE '02: Proceedings of the 17th IEEE international conference on automated software engineering*, page 172, Washington, DC, USA, 2002. IEEE Computer Society.

[61] K. Becker and U. Willie. Communication complexity of group key distribution. In *Proc. 5^{th} Conference on Computers and Communication Security*, pages 1–6, 1998.

[62] D. E. Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. *MITRE technical report, MITRE Corporation, Bedford Massachusetts*, 2997:ref A023 588, 1976.

[63] D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, The MITRE Corporation, Bedford, MA, Jul. 1975.

[64] M. Bellare and P. Rogaway. Entity authentication and key distribution. *CRYPTO 93*, 773, 1994.

[65] M. Bellare and P. Rogaway. Practice-oriented provable security. In *Proc. of First International Workshop on Information Security*, volume 1561, pages 1–15, 1999.

[66] Tim Berners-Lee. The Semantic Web, 2002.

[67] Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. Systematic Design of a Family of Attack-Resistant Protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, June 1993.

[68] D. Boneh. The Decision Diffie-Hellman problem. In *Proc. Third Algorithmic Number Theory Symp*, volume LNCS Vol. 1423, pages 48–63. Springer-Verlag, 1998.

[69] Paulo Borba and Joseph A. Goguen. Refinement of concurrent object oriented programs. In Stephen Goldsack and Stuart Kent, editors, *Formal Methods in Object Technology*. Springer-Verlag, Nov. 1995.

[70] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology - EUROCRYPT'94*, pages 275–286, May 1994.

[71] M. Burmester and Y. G. Desmedt. Efficient and secure conference-key distribution. In *Security Protocols: International Workshop*, pages 119–129, Apr. 1996.

[72] Michael Burrows, Martin Abadi, and Roger Needham. A logic of Authentication. Technical report, Digital Systems Research Center, February 1989. Parts and versions of this material have been presented in many places including ACM Transactions on Computer Systems, 8(1) 18:36, Feb.1990. All references herein are to the SRC Research Report 39 as revised Feb. 22, 1990.

[73] U. Carlsen. Cryptographic protocol flaws. In *Proc. IEEE Computer Security Foundations Workshop VII*, pages 192–200. IEEE Computer Press, June 1994.

[74] Y. Challal, A. Bouabdallah, and H. Bettahar. H2A: Hybrid hash-chaining scheme for adaptive multicast source authentication of media-streaming. *Computers & Security*, 24(1):57–68, 2005.

[75] Lawrence Chung, Brian A. Nixon, and Eric Yu. An approach to building quality into software architecture. In *CASCON '95: Proceedings of the 1995 conference of the centre for advanced studies on collaborative research*, page 13. IBM Press, 1995.

[76] John A. Clark and Jeremy Jacob. A Survey of Authentication Protocol Literature: Version 1.0. University of York, Department of Computer Science, November 1997.

[77] Jane Cleland-Huang. Toward improved traceability of non-functional requirements. In *Proc. 3rd Int'l Workshop on Traceability in Emerging Forms Softw. Eng.: In conj. with (ASE'05)*, pages 14–19, Nov. 2005.

[78] Jane Cleland-Huang, Carl K. Chang, and Mark J. Christensen. Event-based traceability for managing evolutionary change. *IEEE Trans. Softw. Eng.*, 29(9):796–810, Sep. 2003.

[79] Jane Cleland-Huang, Raffaella Settimi, Oussama BenKhadra, Eugenia Berezhan, and Selvia Christina. Goal-centric traceability for managing non-functional requirements. In *Proc. 27th Int'l Conf. on Software Engineering (ICSE'05)*, pages 362–371, May 2005.

[80] Jane Cleland-Huang, Raffaella Settimi, Chuan Duan, and Xuchang Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *Proc. 13th IEEE Int'l Requirements Engineering Conference (RE'05)*, pages 135–144, Aug.–Sep. 2005.

[81] D. Conte de Leon. Formalizing traceability among software work products. Master's thesis, Dept. of Computer Science, University of Idaho, Dec. 2002.

[82] D. Conte de Leon. *Completeness of Implementation Traceability for the Development of High Assurance and Critical Computing Systems*. PhD thesis, Dept. of Computer Science, University of Idaho, Dec. 2006.

[83] D. Conte de Leon and J. Alves-Foss. Hidden implementation dependencies in high assurance and critical computing systems. *IEEE Transactions on Software Engineering*, 32(10):790–811, Oct. 2006.

[84] D. Conte de Leon, J. Alves-Foss, and P. Oman. Implementation-oriented secure architectures, paper st14-01. In *HICSS*, Jan. 2007.

[85] Daniel Conte de Leon and Jim Alves-Foss. Experiments on processing and linking semantically augmented requirement specifications. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, 2004.

[86] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? Here is a new tool that finds some new guessing attacks. In *Proc. Workshop on Issues in the Theory of Security (WITS)*, 2003.

[87] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Comm. ACM*, 31(11):1268–1287, 1988.

[88] J. Dai and J. Alves-Foss. A formal authorization policy model. In *Proc. Software Engineering Research and Applications*, Jun. 2003.

[89] R. I. Damper. Emergence and levels of abstraction. *Int'l Journal of Systems Science*, 31(7):811–818, Jul. 2000.

[90] Mark Denford, John Leaney, and Tim O'Neill. Non-functional refinement of computer based systems architecture. In *ECBS '04: Proceedings of the 11th IEEE international conference and workshop on the Eengineering of computer-based systems*, page 168. IEEE Computer Society, 2004.

[91] Yi Deng, Jiacun Wang, Jeffrey J. P. Tsai, and Konstantin Beznosov. An approach for modeling and analysis of security system architectures. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1099–1119, 2003.

[92] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):553–536, August 1981.

[93] T. Dierks and C. Allen. The TLS protocol Version 1.0. *IETF - Network Working Group*, The Internet Society, RFC 2246, January 1999.

[94] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–652, Nov. 1976.

[95] DoD. *Department of Defense Trusted Computer System Evaluation Criteria*. Department of Defense, 1985.

[96] D. Dolev, S. Even, and R. Karp. On the Security of Ping-Pong Protocols. *Information and Control*, 55(1-3):40–56, 1982.

[97] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.

[98] Alexander Egyed. A scenario-driven approach to traceability. In *Proc. 23rd Int'l Conf. on Software Engineering (ICSE'01)*, pages 123–132, May 2001.

[99] Alexander Egyed. A scenario-driven approach to trace dependency analysis. *IEEE Trans. Softw. Eng.*, 29(2):116–132, 2003.

[100] Alexander Egyed and Paul Grünbacher. Towards understanding implications of trace dependencies among quality requirements. In *Proc. 2nd Int'l Workshop on Traceability in Emerging Forms Softw. Eng.: In conj. with (ASE'03)*, Oct. 2003.

[101] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Why is a security protocol correct? *IEEE Computer Symposium on Security and Privacy*, pages 160–171, 1998.

[102] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2,3):191–230, 1999.

[103] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Mixed Strand Spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, volume 27(2), pages 10–14. IEEE Computer Society Press, June 1999.

[104] M.S. Feather, A.P. Nikora, C.L. Heitmeyer, and N.R. Meade. Workshop on software engineering for high assurance systems. In *25th Int'l Conf. on Software Engineering (ICSE'03)*, Portland, OR, U.S.A., May 2003.

[105] Mariusz A. Fecko and Christopher M. Lott. Improving the requirements engineering process for an electronic clearinghouse. In *Proc. 10th IEEE Int'l Requirements Engineering Conference (RE'02)*, pages 52–60, Essen, Germany, Sep. 2002.

[106] A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partitionable group communication service. In *Proceedings of the 16th Conference on Principles of Distributed Computing*, pages 21–24, Santa Barbara, CA, Aug. 1997.

[107] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Trans. Softw. Eng.*, 20(8):569–578, 1994.

[108] Roberto Fiutem and Giuliano Antoniol. Identifying design-code inconsistencies in object-oriented software: a case study. In *Proc. 14th Int'l Conf. on Software Maintenance (ICSM'98)*, pages 94–102, Nov. 1998.

[109] Xavier Franch and Pere Botella. Putting non-functional requirements into software architecture. In *IWSSD '98: Proceedings of the 9th international workshop on software specification and design*, page 60, Washington, DC, USA, 1998. IEEE Computer Society.

[110] A. Freier, P. Kartion, and P. Kocher. *The SSL Protocol: Version 3.0.* Netscape Communications, InC., Mar. 1996.

[111] Emden R. Gansner, Eleftherios Koutsofios, and Stephen C. North. Drawing graphs with *dot*, Feb. 2002.

[112] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, 30(11):1203–1233, Sep. 2000.

[113] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis, Mathematical Foundations.* Springer-Verlag, Berlin, Germany, 1999.

[114] David Garlan. Style-based refinement for software architecture. In *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops*, pages 72–75, New York, NY, USA, 1996. ACM Press.

[115] Joseph A. Goguen. FOOPS: a functional object-oriented programming system, Feb. 1999.

[116] Joseph A. Goguen. TOOR: a system for tracing object-oriented requirements, Feb. 1999.

[117] Li Gong. Variation on the Themes of Message Freshness and Replay or the Difficulty of Devising Formal Methods to Analyze Cryptographic Protocols. In *Proceedings of the Computer Security Workshop VI*, pages 131–136, Los Alamitos, California, 1993.

[118] Li Gong and Paul Syverson. Fail-stop protocols: An approach to designing secure protocols. In *5th International Working Conference on Dependable Computing for Critical Applicaitons*, pages 44–55, September 1995.

[119] Orlena Gotel and Anthony Finkelstein. An analysis of the requirements traceability problem. In *Proc. First IEEE Int'l Conf. on Requirements Engineering (ICRE'94)*, pages 94–101, Apr. 1994.

[120] Ed Greengrass. Information retrieval: A survey. Technical report, UMBC Center for Architectures for Data-Driven Information Processing (CADIP), Baltimore, MD, U.S.A., Nov. 2000.

[121] D. Greve, M. Wilding, and M. Vanfleet. A separation kernel formal security policy. In *ACL2 Workshop 2003*, 2003.

[122] Joshua D. Guttman and F. Javier Thayer Fábrega. Authentication tests. In *Proceedings, 2000 IEEE Symposium on Secuirty and Privacy*, pages 96–109. IEEE Computer Society Press, May 2000.

[123] Joshua D. Guttman and F. Javier Thayer. Protocol Independence through Disjoint Encryption. *13th IEEE Computer Security Foundations Workshop*, pages 24–34, July 2000.

[124] N. Hanebutte, P. Oman, M. Loosbrock, A. Holland, W. Harrison, and J. Alves-Foss. Software mediators for transparent channel control in unbounded environments. In *Proc. IEEE Systems, Man and Cybernetics Information Assurance Workshop*, pages 30–35, 2005.

[125] Kimberly S. Hanks and John C. Knight. Improving communication of critical domain knowledge in high-consequence software development: an empirical study. In *Proc. 21st Int'l System Safety Conf. (ISSC'03)*, Ottawa, ON, Canada, Aug. 2003.

[126] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). *IETF Network Working Group*, The Internet Society, RFC 2409, November 1998.

[127] W. Scott Harrison, Nadine Hanebutte, Paul Oman, and Jim Alves-Foss. The MILS architecture for a secure global information grid. *Crosstalk: The Journal of Defense Software Engineering*, 2005.

[128] Jane Huffman Hayes, Alex Dekhtyar, and James Osborne. Improving requirements tracing via information retrieval. In *Proc. 11th IEEE Int'l Requirements Engineering Conference (RE'03)*, pages 138–147, Sep. 2003.

[129] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. Softw. Eng.*, 32(01):4–19, Jan. 2006.

[130] James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings, 13th Computer Security Foundations Workshop*, pages 255–268. IEEE Computer Society Press, July 2000.

[131] Mats P. E. Heimdahl and Constance L. Heitmeyer. Formal methods for developing high assurance computer systems: Working group report. In *Proceedings, Second IEEE Workshop on Industrial-Strength Formal Techniques (WIFT'98)*, Oct 1998.

[132] Jean Henrard and Jean-Luc Hainaut. Data dependency elicitation in database reverse engineering. In *Proc. 5th European Conference on Software Maintenance and Reengineering (CSMR'01)*, pages 11–19. IEEE Computer Society, Mar. 2001.

[133] Tzonelih Hwang, Narn-Yih Lee, Chuan-Ming Li, Ming-Yung Ko, and Yung-Hsiang Chen. Two attacks on Neuman-Stubblebine authentication protocols. *Information Processing Letters*, 53(2):103–107, 1995.

[134] Ulla Isaksen, Jonathan P. Bowen, and Nimal Nissanke. System and software safety in critical systems. Technical report, Dept. of Computer Science, Univ. of Reading, Whiteknights, U.K., Dec. 1996.

[135] J. Joy. A content guard for adobe portable document format (pdf). Master's thesis, Dept. of Computer Science, University of Idaho, Aug. 2006.

[136] Jan jürjens. Secrecy-preserving Refinement. In *Proceedings of International Symposium on Formal methods*, volume 2021 of *lncs*, pages 135–152. sv, 2001.

[137] M. Just and S. Vaudenay. Authenticated multi-party key agreement. Technical Report SCS-TR-96-04, Carleton University, Computer Science Department, Ottowa CA, 1996.

[138] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *Advances in Cryptology – ASIACRYPT '96*, pages 36–49, 1996.

[139] P. Karn and W. Simpson. The Photuris session key management protocol. *Internet Engineering Task Force*, RFC 2522, March 1999.

[140] J. Kelsey, B. Schneier, and D. Wagner. Protocol Interactions and the Chosen Protocol Attack. In *Proc. Security Protocols - 5th International Workshop*, pages 91–104. LNCS 1361, 1997.

[141] S. T. Kent, D. Ellis, P. Helinek, K. Sirois, and N. Yuan. Internet routing infrastructure security countermeasures. BBN Report 8173, BBN, January 1997.

[142] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *Proc. 7th ACM Conference on Computer and Communications Security*, Nov. 2000.

[143] Y. Kim, A. Perrig, and G Tsudik. Communication-efficient group key agreement. In *Proc. of IFIP SEC 2001*, Jun. 2001.

[144] Y. Kim, A. Perrig, and G. Tsudik. Group key agreement efficient in communication. *IEEE Transactions on Computers*, 53(7):905–921, Jul. 2004.

[145] Y. Kim, A. Perrig, and G. Tsudik. Tree-based group key agreement. *ACM Transactions on Information and System Security*, 7(1):60–96, Feb. 2004.

[146] John C. Knight, Elisabeth A. Strunk, William S. Greenwell, and Kimberly S. Wasson. Specification and analysis of data for safety-critical systems. In *Proc. 22nd Int'l System Safety Conf. (ISSC'04)*, Providence, RI, U.S.A., Aug. 2004.

[147] J. Kohl and C. Neuman. The Kerberos network authentication service (v5). Network working group, RFC 1510, September 1993.

[148] L. Lamport. Time clocks and the ordering of events in a distributed system. *Communications of ACM*, 21(7):558–565, January 1978.

[149] Jean Claude Laprie. *Dependability: Basic Concepts and Terminology.* Springer-Verlag, Berlin, Germany, 1992.

[150] H. Lee. *Securing Mobile Agents through Evaluation of Encrypted Functionss.* PhD thesis, Dept. of Computer Science, University of Idaho, Aug. 2002.

[151] Hyungjick Lee, Jim Alves-Foss, and Scott Harrison. Securing mobile agents through evaluation of encrypted functions. *Web Intelligence and Agent Systems*, 2(1):1–19, 2004.

[152] Nacny G. Leveson, Maxime de Villepin, Mirna Daouk, John Bellingham, Jayakanth Srinivasan, Natasha Neogi, Ed Bachelder, Nadince Pilon, and Geraldine Flynn. A safety and human-centered approach to developing new air traffic management tools. Technical report, Aeronautics and Astronautics Department, MIT, and Eurocontrol Experimental Centre, Dec. 2001.

[153] Nancy G. Leveson. Completeness in formal specification language design for process-control systems. In *3rd Workshop on Formal Methods in Software Practice*, pages 75–87, Portland, OR, U.S.A., Mar. 2000. ACM Press.

[154] Nancy G. Leveson. Intent specifications: An approach to building human-centered specifications. *IEEE Trans. Softw. Eng.*, 26(01):15–35, Jan. 2000.

[155] Nancy G. Leveson. Evaluating accident models using recent aerospace accidents, part one: Event-based models. Technical report, Software Engineering Research Laboratory, Massachusetts Inst. of Tech., Cambridge, MA, U.S.A., Jun. 2001.

[156] Nancy G. Leveson. The role of software in spacecraft accidents. *AIAA J. of Spacecraft and Rockets*, 41(4):564–575, Jul. 2004.

[157] Nancy G. Leveson. A systems-theoretic approach to safety in software intensive systems. *IEEE Trans. Dependable and Secure Comput.*, 1(1):66–86, Jan. 2004.

[158] Nancy G. Leveson. *System Safety Engineering: Back to the Future.* Draft of Book, Jun. 2006.

[159] Nancy G. Leveson, Mirna Daouk, Nicolas Dulac, and Karen Marais. A systems theoretic approach to safety engineering. Technical report, Dept. of Aeronautics and Astronautics, Massachusetts Inst. of Tech., Cambridge, MA, U.S.A., Oct. 2003.

[160] Nancy G. Leveson, Mats Per Erik Heimdahl, and Jon Damon Reese. Designing specification languages for process control systems: Lessons learned and steps to the future. In *7th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 127–145, Toulouse, France, Sep. 1999.

[161] Azriel Levy. *Basic Set Theory.* Springer-Verlag, Berlin, Germany, 1999.

[162] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer-Verlag, 1996. Also in Software Concepts and Tools, 17:93-102, 1996.

[163] Gavin Lowe. Some new attacks on cryptographic protocols. In *Proceedings of 9th Computer Security Foundations Workshop*, pages 162–170. IEEE Computer Press, March 1996.

[164] Gavin Lowe. A heirarchy of authentication specifications. In *10th Computer Security Foundations Workshop Proceedings*, pages 31–43, 1997.

[165] Robyn R. Lutz. Analyzing software requirements errors in safety-critical embedded systems. Technical report, Iowa State Univ. of Science and Technology, Dept. of Computer Science, Ames, IA, U.S.A., Aug. 1992.

[166] Robyn R. Lutz. Analyzing software requirements errors in safety-critical embedded systems. In *1st IEEE Int'l Symp. on Requirements Engineering (RE'93)*, pages 126–133, San Diego, CA, U.S.A., Jan. 1993.

[167] Robyn R. Lutz and Ines Carmen Mikulski. Empirical analysis of safety-related anomalies during operations. *IEEE Trans. Softw. Eng.*, 30(3):172–180, Mar. 2004.

[168] Jonathan I. Maletic, Ethan V. Munson, Andrian Marcus, and Tien N. Nguyen. Using a hypertext model for traceability link conformance analysis. In *Proc. 2nd Int'l Workshop on Traceability in Emerging Forms Softw. Eng.: In conj. with (ASE'03)*, Oct. 2003.

[169] S. Malladi. A general scheme to prevent replay attacks on security protocols. Master's thesis, Dept. of Computer Science, University of Idaho, Dec. 2002.

[170] S. Malladi. *Formal Analysis and Verification of Password Protocols.* PhD thesis, Dept. of Computer Science, University of Idaho, Jun. 2004.

[171] S. Malladi and J. Alves-Foss. How to prevent type-flaw guessing attacks on password protocols. In *Proc. Foundations of Computer Security*, Jun. 2003.

[172] Sreekanth Malladi, J. Alves-Foss, and Robert B. Heckendorn. On preventing replay attacks on security protocols. In *Proc. International Conference on Security and Management*, pages 77–83, Jun. 2002.

[173] Sreekanth Malladi, Jim Alves-Foss, and Sreenivas Malladi. Preventing guessing attacks using fingerprint biometrics. In *Proc. International Conference on Security and Management*, Jun. 2002.

[174] Sreekanth Malladi, Jim Alves-Foss, and Sreenivas Malladi. What are multi-protocol guessing attacks and how to prevent them. In *Proc. 7th International Workshop on Enterprise Security*, Jun. 2002.

[175] Evan Mamas and Kostas Kontogiannis. Towards portable source code representations using XML. In *Proc. 7th Working Conf. on Reverse Engineering (WCRE'00)*, pages 172–182, Brisbane, Australia, Nov. 2000.

[176] D. Manz. A network simulator for group key management algorithms. Master's thesis, Dept. of Computer Science, University of Idaho, Dec. 2005.

[177] D. Manz, J. Alves-Foss, and S. Zheng. A network simulator for group key management algorithms. *Journal Information Assurance and Security*, 2(4), 2007 in press.

[178] Adrian Marcus and Jonathan I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proc. 25th Int'l Conf. on Software Engineering (ICSE'03)*, May 2003.

[179] W. Martin, P. White, F. S. Taylor, and A. Goldberg. Formal construction of the mathematicaly analyzed separation kernel. In *Proc. of the* 15^{th} *International Conference on Automated Software Engineering*, pages 133–141, 2001.

[180] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet Security Association and Key Management Protocol (ISAKMP). IETF Network Working Group RFC 2408, November 1998.

[181] D. McCullough. Foundations of Ulysses: The theory of security. Technical Report RADC-TR-87-222, Odyssey Research Associates, Inc., Jul. 1988.

[182] D. McCullough. Noninterference and the composability of security properties. In *Proc. IEEE Symposium on Security and Privacy*, pages 177–187, 1988.

[183] D. McCullough. Noninterference and the composability of security properties. In *Proc. IEEE symposium on security and privacy*, pages 177–187, 1988.

[184] J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, Jan. 1996.

[185] John McLean and Constance Heitmeyer. High assurance computer systems: A research agenda. Technical report, Center for High Assurance Computer Systems, Naval Research Laboratory, Washington, DC, U.S.A., 1995.

[186] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[187] C.A. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. *ESORICS 96, LNCS 1146*, pages 351–364, 1996.

[188] Catherine Meadows. A model of computation for the NRL Protocol Analyzer. In *Proceedings of the 7th Computer Security Foundations Workshop*, pages 84–89, June 1994.

[189] Catherine Meadows. Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer. In *Proceedings, 1999 IEEE Symposium on Security and Privacy*, pages 48–61. IEEE Computer Society Press, May 1999.

[190] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[191] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, USA, 5th edition, 1996.

[192] D. Meyers. An attribute grammar for alert aggregation in intrusion detection systems. Master's thesis, Dept. of Computer Science, University of Idaho, May 2007.

[193] Jeff Michaud, Margaret-Anne Storey, and Hausi A. Müller. Integrating information sources for visualizing Java programs. In *17th Int'l Conf. on Software Maintenance (ICSM'01)*, pages 250–259, Florence, Italy, Nov. 2001.

[194] C. Mitchell. Limitations of Challenge-Response Entity Authentication. *Electronic Letters*, 25(17):1195–1196, August 1989.

[195] Francesmary Modugno, Nancy G. Leveson, Jon Damon Reese, Kurt Partridge, and Sean D. Sandys. Integrated safety analysis of requirements specifications. In *Proc. 3rd IEEE Int'l Symp. on Requirements Engineering (RE'97)*, pages 148–159, Annapolis, MD, U.S.A., Jan. 1997.

[196] M. Moriconi, Xiaolei Qian, R. A. Riemenschneider, and Li Gong. Secure software architectures. In *SP '97: Proceedings of the 1997 IEEE symposium on security and privacy*, page 84, Washington, DC, USA, 1997. IEEE Computer Society.

[197] Mark Moriconi, Xiaolei Qian, and R. A. Riemenschneider. Correct architecture refinement. *IEEE Transactions on Software Engineering*, 21(4):356–3, 1995.

[198] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. In *Proc. of the 3rd. Int'l Semantic Web Conference (ISWC 2004)*, pages 549–563. Springer-Verlag, Nov. 2004.

[199] Francisco Naishtat. *Lógica para Computación*. Editorial Universitaria de Buenos Aires (EUDEBA), Buenos Aires, Argentina, 1986.

[200] National Transportation Safety Board. Controlled flight into terrain, Korean Air Flight 801, Boeing 747-300 HL7468, Nimitz Hill, Guam, 06 August 1997. Aircraft Accident Report NTSB/AAR-00/01, NTSB, Washington, DC, U.S.A., 2000.

[201] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[202] Christian Nentwich, Lucia Capra, Wolfgang Emmerich, and Anthony Finkelstein. Xlinkit: A consistency checking and smart link generation service. *ACM Trans. Internet Tech.*, 2(2):151–185, May 2002.

[203] Christian Nentwich, Wolfgang Emmerich, and Anthony Finkelstein. Static consistency checking for distributed specifications. In *Proc. 16th Int'l Conf. on Automated Software Eng. (ASE'01)*, page 115, Nov. 2001.

[204] Christian Nentwich, Wolfgang Emmerich, and Anthony Finkelstein. Consistency management with repair actions. In *Proc. 25th Int'l Conf. on Software Engineering (ICSE'03)*, pages 455–464, May 2003.

[205] Christian Nentwich, Wolfgang Emmerich, Anthony Finkelstein, and Ernst Ellmer. Flexible consistency checking. *ACM Trans. Softw. Eng. and Methodology*, 12(1):28–63, Jan. 2003.

[206] B. Clifford Neuman and Stuart G. Stubblebine. A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10-14, April 1993.

[207] K. Nyberg and L. R. Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8(1):27–37, Winter 1995.

[208] Patrick O'Connell. The Idaho Partitioning Machine: A MILS partitioning kernel model in ACL2. Master's thesis, Dept. Computer Science, University of Idaho, Dec. 2003.

[209] P. Oman, A. Krings, D. Conte de Leon, and J. Alves-Foss. Analyzing the security and survivability of real-time control systems. In *Proc. IEEE Systems, Man and Cybernetics Information Assurance Workshop*, pages 342–349, Jun. 2004.

[210] The Open Group. *The Partitioning Kernel Protection Profile*, Jun. 2003. Draft under review.

[211] D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[212] A. Perrig. Efficient collaborative key management protocols for secure autonomous group communication. In *Proc. CrypTEC '99*, pages 192–202, 1999.

[213] B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic security of reactive systems. *eNTCS*, 32, 1999.

[214] Jan Philipps and Bernhard Rumpe. Refinement of pipe-and-filter architectures. In *FM '99: Proceedings of the wold congress on formal methods in the development of computing systems-Volume I*, pages 96–115, London, UK, 1999. Springer-Verlag.

[215] Francisco A. C. Pinheiro and Joseph A. Goguen. An object-oriented tool for tracing requirements. *IEEE Softw.*, 13(2):52–64, Mar. 1996.

[216] Klaus Pohl. Pro-art: Enabling requirements pre-traceability. In *Proc. 2nd IEEE Int'l Conf. on Requirements Engineering (ICRE'96)*, pages 76–84, Apr. 1996.

[217] Klaus Pohl. *Process-Centered Requirements Engineering.* John Willey and Sons, Taunton, U.K., 1996.

[218] Karl R. Popper and John C. Eccles. *The Self and Its Brain.* Springer-Verlag, Berlin, Germany, 1977.

[219] Balasubramaniam Ramesh and Matthias Jarke. Toward reference models of requirements traceability. *IEEE Trans. Softw. Eng.*, 27(01):58–93, Jan. 2001.

[220] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[221] J. Robinson. A high-assurance multi-level secure file server. Master's thesis, Dept. of Computer Science, University of Idaho, Dec. 2006.

[222] J. Robinson and J. Alves-Foss. A high assurance MLS file server. *ACM Operating Systems Review*, 41(1):45–53, Jan. 2007.

[223] J. Robinson, W.S. Harrison, N. Hanebutte, P. Oman, and J. Alves-Foss. Implementing middleware for content filtering and information flow control. In *Proc. Computer Security Architecture Workshop*, pages 47–53, Nov. 2007.

[224] H. L. Rogers. An overview of the CANEWARE program. In *Proc. 10th NIST-NCSC National Computer Security Conference*, pages 172–174, 1987.

[225] Nelson S. Rosa, George R. R. Justo, and P. R. F. Cunha. Incorporating non-functional requirements into software architectures. In *IPDPS '00: Proceedings of the 15 IPDPS 2000 workshops on parallel and distributed processing*, pages 1009–1018, London, UK, 2000. Springer-Verlag.

[226] Nelson S. Rosa, George R. R. Justo, and Paulo R. F. Cunha. A framework for building non-functional software architectures. In *SAC '01: Proceedings of the 2001 ACM symposium on applied computing*, pages 141–147, New York, NY, USA, 2001. ACM Press.

[227] B. Rossebo, P. Oman, J. Alves-Foss, R. Blue, and P. Jaszkowiak. Using Spark-Ada to model and verify a MILS message router. In *Proc. Int'l Symposium on Secure Software Enginneering*, Mar. 2006.

[228] RTCA. Software considerations in airborne systems and equipment certification. Technical Report DO-178b/ED-12B, RTCA, Inc., 1993. URL: http://www.stsc.hill.af.mil/crosstalk/1998/10/schad.asp.

[229] RTCA. Requirements specification for avionics computer resource. Technical Report DO-255, RTCA, Inc., 2000.

[230] J. Rushby and B. Randell. A distributed secure system. *IEEE Computer*, 16(7):55–67, 1983.

[231] J. M. Rushby. Proof of separability: A verification technique for a class of security kernels. *Proc. International Symposium on Programming, Lecture Notes in Computer Science*, 137:352–367, 1982.

[232] J.M Rushby. Design and verification of secure systems. In *Proc. ACM Symposium on Operating System Principles*, volume 15, pages 12–21, 1981.

[233] John Rushby. Critical systems properties: Survey and taxonomy. Technical Report CSL-93-01, Center for High Assurance Computer Systems, Naval Research Laboratory, Washington, DC, U.S.A., May 1994. Rev. Feb. 1994.

[234] Peter Ryan and Steve Schneider. *Modelling and Analysis of Security Protcols*. Addison-Wesley, An imprint of Pearson education Limited, 128 Long Acre, London, WC2E9AN, 2001.

[235] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communication*, 21(1), January 2003.

[236] SAE Embedded Computing Systems Committee. Architecture analysis & design language (aadl). Technical Report 1.0, Society of Automotive Engineering Int'l., Nov. 2004.

[237] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceeding of the IEEE*, 63(9):1278–1308, Sept. 1975.

[238] Scott A. Selberg and Mark A. Austin. Requirements engineering and the Semantic Web. Technical Report TR 2003-20, The Institute for Systems Research, College Park, MD, U.S.A., 2003.

[239] Raffaella Settimi, Jane Cleland-Huang, Oussama BenKhadra, Jigar Mody, Wiktor Lukasik, and Chris DePalma. Supporting software evolution through dynamically retrieving traces to UML artifacts. In *Proc. 7th Int'l Workshop on Principles of Software Evolution: in conj. with (RE'04)*, pages 49–54, Sep. 2002.

[240] C.E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28-4:656 – 715, October 1949.

[241] Susanne A. Sherba, Kenneth M. Anderson, and Maha Faisal. A framework for mapping traceability relationships. In *Proc. 2nd Int'l Workshop on Traceability in Emerging Forms Softw. Eng.: In conj. with (ASE'03)*, Oct. 2003.

[242] G. J. Simmons. How to (Selectively) Broadcast a Secret. In *Proceedings of the 1985 IEEE Computer Society Symposium on Security and Privacy*, pages 108–113, 1985.

[243] Raymond Smullyan. *Theory of Formal Systems*. Princeton Univ. Press, Princeton, N.J., 1961.

[244] Software Engineering Institute. Capability maturity model integration for systems engineering, software engineering, integrated product and process development, and supplier sourcing. Technical Report CMU/SEI-2002-TR-012, Carnegie Mellon Univ., Pittsburgh, PA, U.S.A., Mar. 2002.

[245] J. Son. *Covert Timing Analysis in MLS Real-Time Systems*. PhD thesis, Dept. of Computer Science, University of Idaho, Jun. 2008.

[246] J. Son and J. Alves-Foss. Covert timing channel analysis of rate monotonic real-time scheduling algorithm in mls systems. In *Proc. IEEE Information Assurance Workshop*, pages 361–368, Jun. 2006.

[247] J. Son and J. Alves-Foss. High level specification of non-interference security policies in partitioned MLS systems. In *Proc. IASTED International Conf. on Communication, Network and Information Security (CNIS 2007)*, Sep. 2007.

[248] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. Technical report, Information Sciences Institute, Jan. 1999.

[249] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to groups. In *Third ACM Conference on Computer and Communications Security*, pages 31–37. ACM Press, Mar. 1996.

[250] Rita. C Summers. *Secure computing : Threats and Safeguards*. McGraw-Hill, Hightstown, NJ, USA, 1997.

[251] Jing Sun, Jin Song Dong, Jing Liu, and Hai Wang. A formal object approach to the design of ZML. *Annals of Software Eng.*, 13(1-4):329–356, 2002.

[252] Paul Syverson. A taxonomy of replay attacks. In *Proceedings of the Computer Security Foundations Workshop (CSFW97)*, pages 187–191, June 1994.

[253] Paul F. Syverson. On Key Distribution Protocols for Repeated Authentication. *Operating Systems Review*, 27(4):24:30, October 1993.

[254] C. Taylor. *Techniques for the Survivability of Critical Computer Systems.* PhD thesis, Dept. of Computer Science, University of Idaho, Jun. 2004.

[255] Carol Taylor, Jim Alves-Foss, and Bob Rinker. Merging safety and assurance: The process of dual certification for software. In *Proc. of the Systems and Software Technology Conference*, Salt Lake City, UT, U.S.A., Apr. 2002.

[256] Thomas Tilley. Formal concept analysis applications to requirements engineering and design. Dissertation, Univ. of Queensland, Brisbane, Australia, Nov. 2003.

[257] Thomas Tilley. Towards an FCA based tool for visualizing formal specifications. In *Using Conceptual Structures: Contributions to ICCS 2003*, pages 227–240, Los Angeles, CA, U.S.A., 2003.

[258] Thomas Tilley, Richard Cole, Peter Becker, and Peter Eklund. A survey of formal concept analysis support for software engineering activities. In *Proc. First Int'l Conf. on Formal Concept Analysis (IFFCA'03)*, Feb. 2003.

[259] William Van Lepthien and Kenneth M. Anderson. Unifying structure, behavior, and data with themis types and templates. In *Proc. 15th ACM Conf. on Hypertext and Hypermedia (HT'04)*, pages 256–265, Aug. 2004.

[260] L. Wahsheh. *Security Policy Design and Implementation in High Assurance Computer Systems.* PhD thesis, Dept. of Computer Science, University of Idaho, Jun. 2008.

[261] L.A. Wahsheh and J. Alves-Foss. Specifying and enforcing a multi-policy paradigm for high assurance embedded systems. *International Journal of High Speed Networks*, 15(3):315–327, Oct. 2006.

[262] L.A. Wahsheh and J. Alves-Foss. Policy-based security for wireless components in high assurance systems. *Journal of Computer Science*, 3(9):727–739, 2007.

[263] L.A. Wahsheh and J. Alves-Foss. Using policy enforcement graphs in a separation-based high assurance architecture. In *Proc. IEEE International Conference on Information Reuse and Integration*, pages 183–189, Aug. 2007.

[264] L.A. Wahsheh and J. Alves-Foss. Security policy development: Towards a life-cycle and logic-based verification model. *American Journal of Applied Sciences*, 5(9):1117–1126, 2008.

[265] L.A. Wahsheh, D. Conte de Leon, and J. Alves-Foss. Formal verification and visualization of security policies. *Journal of Computers*, 3(6), 2008.

[266] B. Wang. Possibilistic information flow analysis and formal verification of multiple single-level secure execution monitoring mechanisms for high assurance systems. Master's thesis, Dept. of Computer Science, University of Idaho, May 2006.

[267] B. Wang and J. Alves-Foss. An MSLS-EMM for enforcing confidentiality in malicious environments. In *IASTED International Conf. on Communication, Network and Information Security (CNIS 2006)*, pages 126–131, Oct. 2006.

[268] Kathryn Anne Weiss, Nancy G. Leveson, Kristina Lundqvist, Nida Farid, and Margaret Stringfellow. An analysis of causation in aerospace accidents. In *Proc. Digital Avionics Systems Conf.(DASC-2001)*, pages 137–147. AIAA, IEEE, Oct. 2001.

[269] C. Weissman. BLACKER: Security for the DDN examples of A1 security engineering trades. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 286–292, 1992.

[270] P. White, W. M. Van Fleet, and C. Dailey. High assurance architecture via separation kernel. Draft, Oct. 2000.

[271] Andreas Winter, Bernt Kullbach, and Volker Riediger. An overview of the GXL graph exchange language. *Lecture Notes in Comput. Science*, 2269:324–336, May 2002.

[272] T.Y.C. Woo and S. S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 28(3):24–37, 1994.

[273] CCITT Recommendation X.400. Message handling system and service overview. Technical report, CCITT, November 1996.

[274] D. Yang. A threat-scenario-driven modeling approach to the mmr. Master's thesis, Dept. of Computer Science, University of Idaho, Aug. 2005.

[275] T. Ylonen. SSH– secure login connections over the Internet. In *the Sixth USENIX Unix Security Symposium*, pages 37–42, July 1996.

[276] Zhifeng Yu and Václav Rajlich. Hidden dependencies in program comprehension and change propagation. In *Proc. 9th Int'l Workshop on Program Comprehension (IWPC'01)*, pages 293–299, Washington, DC, U.S.A., 2001. IEEE Computer Society.

[277] A. Zakinthinos. *On The Composition Of Security Properties.* PhD thesis, University of Toronto, Mar. 1996.

[278] Edward N. Zalta, editor. *The Stanford Encyclopedia of Philosophy*, chapter Emergent Properties. The Metaphysics Research Lab Center for the Study of Language and Information, Stanford, CA, U.S.A., Mar. 2006.

[279] Lantian Zheng and Andrew C. Myers. End-to-end availability policies and noninterference. In *CSFW '05: Proceedings of 18th IEEE computer security foundation workshop*, 2005.

[280] S. Zheng. *A Communication-Computation Efficient Group Key Algorithm for Large and Dynamic Groups.* PhD thesis, Dept. of Computer Science, University of Idaho, Aug. 2006.

[281] S. Zheng, J. Alves-Foss, and S. Lee. The effect of rebalancing on the performance of a group key agreement protocol. In *Annual IEEE Conference on Local Computer Networks*, pages 983–989, Nov. 2006.

[282] S. Zheng, D. Manz, and J. Alves-Foss. A communication-computation efficient group key algorithm for large and dynamic groups. *Journal of Computer Networks*, 51(1):69–93, 2007.

[283] S. Zheng, D. Manz, J. Alves-Foss, and Y. Chen. Security and performance of group key agreement protocols. In *Proc. IASTED Networks and Communication Systems*, pages 321–327, Mar. 2006.

[284] Shanyu Zheng, Jim Alves-Foss, and Stephen Lee. Exploring average performance of group key management algorithms over multiple operations. In *Proc. IASTED International Conference on Communications, Internet, and Information Technology (CITT 2005)*, 2005.

[285] J. Zhou and J. Alves-Foss. Security policy refinement and enforcement in secure computer systems design. *Journal of Computer Security*, 16(2):107–131, 2008.

[286] J. Zhou and D. Gollmann. Evidence and non-repudiation. *Journal of Network and Computer Applications*, 20(3):267–281, 1997.

[287] Jie Zhou and Jim Alves-Foss. Architecture-based refinements for secure computer systems design. In *Proc. Policy, Security and Trust*, 2006.

[288] Jie Zhou and Jim Alves-Foss. Architecture-based refinements for secure computer systems design. In *PST '06: Proceedings of 4th international conference on privacy, security, and trust*, pages 89–102, October 2006.

Abstracts of Publications

The following is an annotated bibliography of the papers, Ph.D. dissertations and Masters Thesis related to this contract. Although these documents we addressed in earlier sections of this report, we have included this bibliography, with abstracts to give the reader a fuller understanding of the material published here.

Bibliography

Al-Muhaitheef02a

[Al-Muhaitheef02a] A. Al-Muhaitheef. *The Firewall Mobile Customs Agents: A Distributed Firewall Architecture.* PhD thesis, Dept. of Computer Science, University of Idaho, Aug. 2002.

Abstract: The revolution of modern networking necessitates many new security methods to protect our communications from intruders. For example, users of the Internet employ encryption methods to protect their communications from spoofing or modification, and use tunneling techniques to hide their identities, while network system administrators protect their local networks by routers and firewalls to filter the communication passing through. Using different types of security tools in network communication may result in some conflicts. For example, using an encryption method to protect the integrity and privacy of data may prevent a firewall from inspecting incoming or outgoing data from the local network. Combining these two methods will result in a strong security system that protects the local network and the privacy of the connection. However, it may compromise some security features, like the security of the encryption key from users other than the encryption channel participants. In this dissertation, we are defining a new approach to combining these two techniques by handling a virtual private network through a firewall by a mobile agent; using an analogy to the real life example of a customs agent inspector. This agent will work at the end point of the connection for inspection as a delegate of the firewall and by its signature approve legitimate packets to pass the firewall without inspection.

Alves-Foss02a

[Alves-Foss02a] J. Alves-Foss, D. Conte de Leon, and P. Oman. Experiments in the use of XML to enhance traceability between object-oriented design specs. and source code. In *Proc. Of the 35th Hawaii Intl. Conf. On System Sciences*, pages 3959 – 3966, 2002.

Abstract: In this paper we explain how we implemented traceability between a UML design specification and its implementing source code using XML technologies. In our linking framework an XMI file represents a detailed-design specification and a JavaML file represents its source code. These XML-derivative representations were linked using another XML file, an Xlink link-base, containing our linking information. This

link-base states which portions of the source code implement which portions of a design specification and vice-versa. We also rendered those links to an HTML file using XSL and traversed from our design specification to its implementing source code. This is the first step in our traceability endeavors where we aim to achieve total traceability among software life-cycle deliverables form requirements to source code.

Alves-Foss04a

[Alves-Foss04a] J. Alves-Foss and C. Taylor. An analysis of the GWV security policy. In *ACL2 Workshop 2004*, 2004.

Abstract: The use of formal models of security policies are required for high assurance security systems. One benefit of formal methods is that it allows for a precise presentation of items, allowing for analysis by others and subsequent discussion. In this paper we examine the presentation and use of the formal security policy developed in ACL2 as presented by Greve, Wilding and Vanfleet in 2003. We found that the ACL2 model and corresponding textual description left some points ambiguous. We clarify these points in this paper.

Alves-Foss04b

[Alves-Foss04b] Jim Alves-Foss, Carol Taylor, and Paul W. Oman. A multi-layered approach to security in high assurance systems. In *Proc. Hawaii Systems Sciences Conference*, 2004.

Abstract: Past efforts at designing and implementing ultra high assurance systems for government security and safety have centered on the concept of a monolithic security kernel responsible for a system-wide security policy. This approach leads to inflexible, overly complex operating systems that are too large to evaluate at the highest assurance levels (e. G., Common Criteria EAL 5 and above). We describe a new multilayered approach to the design and verification of embedded trustworthy systems that is currently being used in the implementation of real time, embedded applications. The framework supports multiple levels of safety and multiple levels of security, based on the principle of creating separate layers of responsibility and control, with each layer responsible for enforcing its own security policy.

Alves-Foss06a

[Alves-Foss06a] J. Alves-Foss, W.S. Harrison, P. Oman, and C. Taylor. The MILS architecture for high assurance embedded systems. *International Journal of Embedded Systems*, 2(3/4):239–247, 2006.

Abstract: High-assurance systems require a level of rigor, in both design and analysis, not typical of conventional systems. This paper provides an overview of the Multiple Independent Levels of Security and Safety (MILS) approach to high-assurance system design for security and safety critical embedded systems. MILS enables the development of a system using manageable units, each of which can be analysed separately, avoiding costly analysis required of more conventional designs. MILS is particularly well suited to embedded systems that must provide guaranteed safety or security properties.

ConteDeLeon02b

[ConteDeLeon02b] D. Conte de Leon. Formalizing traceability among software work products. Master's thesis, Dept. of Computer Science, University of Idaho, Dec. 2002.

Abstract: The software development process generates a set of software work products. Those software work products together form a model of the system being developed and its justification, therefore they need to be related. The purpose of traceability is to create and maintain those relationships among software work products in order to help software practitioners gain better understanding of the system, which improves overall system quality. Traceability is one of the most difficult and current problems in software engineering. It is not a new topic in the research community. However, traceability has not reached a maturity state in most software organizations or software development projects. In addition, software contracts usually require compliance with development standards such as ISO 12207 or IEEE / IEC 12207. Those standards mandate traceability as a property of a software project. However, there are currently no means to prove that a software project holds the required traceability properties. In this work, we aim to solve this problem. In order to prove a property we need first a formal model and a formal definition of that property. We formalized traceability among software work product sections using concepts of set theory and graph theory. We developed a framework where traceability can be implemented with any desired level of granularity. We formally defined three new concepts: object domain, implementation domain, and implementation traceable software project. We developed a formal link specification language, called TraceML, along with its formal syntax and semantics. We conducted two proof-of-concept experiments on the implementation of our framework and formal traceability, which demonstrate that it is possible to use our approach to improve traceability in the software engineering process.

ConteDeLeon06a

[ConteDeLeon06a] D. Conte de Leon. *Completeness of Implementation Traceability for the Development of High Assurance and Critical Computing Systems.* PhD thesis, Dept. of Computer Science, University of Idaho, Dec. 2006.

Abstract: High assurance and critical computing systems require compelling evidence that they satisfy certain critical properties while achieving their functional objectives. Such compelling evidence must be collected and assembled during the analysis, development, and maintenance processes in order to enable assurance of correctness during the evaluation, certification, and auditing processes. During these processes, a myriad of artifacts and deliverables are created, evolved, and managed, which are usually referred to as work products. Traceability of work product sections is the ability of a stakeholder to manually or mechanically describe and navigate relationships between uniquely identifiable and addressable sections of work products. Catastrophic failures leading to life, environmental, and other great damages and losses, can arise from unfounded assumptions of independence between system requirements, constraints, and components (work product sections), which may stem from misunderstanding and miscommunication between system model stakeholders. Ensuring effective and complete traceability between work product sections may help to avoid misunderstandings among stakeholders and hence avoid unexpected interactions between work product sections, which could lead to critical or catastrophic failures. In addition, traceability is the centerpiece of the assurance evidence and the vehicle that enables the evaluation, certification, and auditing processes mandated for high assurance and critical computing systems. In this dissertation, firstly, I construct a formal theory of work product sections and their associated traceability relations. Secondly, I analyze and formalize the semantics of the partial implementation traceability relation. Thirdly, I describe a formal technique for the discovery of potential causes of critical failures. Fourthly, I introduce a technique for the mechanical completion of a given set of partial implementation traceability links. Fifthly, I introduce a new implementation-oriented and holistic methodology for the development of high assurance and critical computing systems. I demonstrate how this approach to traceability with its associated techniques may be used to help stakeholders discover potential causes of critical failures in high assurance and critical computing systems by applying the techniques to five case studies. In addition, throughout this dissertation, I present a prototype expert system named SyModEx, which implements the formal pproach and some of the techniques described in this work.

ConteDeLeon06b

[ConteDeLeon06b] D. Conte de Leon and J. Alves-Foss. Hidden implementation dependencies in high assurance and critical computing systems. *IEEE Transactions on Software Engineering*, 32(10):790–811, Oct. 2006.

Abstract: Critical and catastrophic failures in high assurance and critical computing systems can arise from unfounded assumptions of independence between system components, requirements, and constraints (work product sections), which can stem from misunderstandings and miscommunication between system engineers, managers, and operators and from inadequate or incomplete traceability between system work products. In this article, we propose a formal framework for the effective implementation of traceability between work product sections along with a technique for discovering potential causes of critical failures in high assurance and critical computing system models. We introduce a new abstraction of interrelated work product sections called implementation meta-work product and describe how our technique finds these meta-work products. We also demonstrate how this technique can be used to help analysts discover potential causes of safety-related errors in high assurance and critical computing systems by applying it to one case study of a known

critical error and to one case study where we anticipate potential safety hazards.

ConteDeLeon07a

[ConteDeLeon07a] D. Conte de Leon, J. Alves-Foss, and P. Oman. Implementation-oriented secure architectures, paper st14-01. In *HICSS*, Jan. 2007.

Abstract: We propose a framework for constructing secure systems at the architectural level. This framework is composed of an implementation-oriented formalization of a system's architecture, which we call the formal implementation model, along with a method for the construction of a system based on elementary analysis, implementation, and synthesis steps. Using this framework, security vulnerabilities can be avoided by constraining the architecture of a system to those architectures that can be rigorously argued to implement all corresponding functional and security requirements, and no other. Furthermore, the framework enables the verification and validation of system correctness by enforcing traceability of final system components to their corresponding design, architecture, and requirement work products

Corin03a

[Corin03a] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? Here is a new tool that finds some new guessing attacks. In *Proc. Workshop on Issues in the Theory of Security (WITS)*, 2003.

Abstract: If a protocol is implemented using a poor password, then the password can be guessed and verified from the messages in the protocol run. This is termed as a guessing attack. Published design and analysis efforts always lacked a general definition for guessing attacks. Further, they never considered possible type-aws in the protocol runs or using messages from other protocols. In this paper, we provide a simple and general definition for guessing attacks. We explain how we implemented our definition in a tool based on constraint solving. Finally, we demonstrate some new guessing attacks that use type-flaws and multiple protocols which we found using our tool.

Dai03a

[Dai03a] J. Dai and J. Alves-Foss. A formal authorization policy model. In *Proc. Software Engineering Research and Applications*, Jun. 2003.

Abstract: This paper presents a formal model that interprets authorization policy behaviors. The model establishes a connection of applying authorization policies on an administration domain with dissecting the domain into the authorized, denied, and undefined divisions. This connection enables us to analyze authorization policy development problems such as policy merge, inconsistency, ambiguity, and redundancy by examining the domain elements mapped into each of the divisions. In addition, three distinct authorization values are assigned to the divisions based on the permission of access control, and are used to calculate partition index

of each rule or policy for measurement purpose. The entire measurable model provides a method to analyze and develop correct and conflict free authorization policies.

Hanebutte05a

[Hanebutte05a] N. Hanebutte, P. Oman, M. Loosbrock, A. Holland, W. Harrison, and J. Alves-Foss. Software mediators for transparent channel control in unbounded environments. In *Proc. IEEE Systems, Man and Cybernetics Information Assurance Workshop*, pages 30–35, 2005.

Abstract: Establishing verifiably secure communications is a daunting task, especially in unbounded computing networks such as the Internet and the global information grid. The multiple independent levels of security (MILS) architecture has been developed to facilitate this task. Wrappers, filters and mediators, both hardware and software, have been proposed as MILS mechanisms to enforce communication security policies such as data isolation and sanitation. This paper describes two experimental projects showing how software mediators can be implemented using CORBA in two different environments: a standard Unix TCP/IP network with multiple workstations, and a single board computer running the integrity operating system with a separation kernel supporting multiple isolated execution environments. The first example shows how protocol mediators can enforce communication-related security policies on standard networks, while the second shows that same functionality implemented on a MILS-based architecture. The projects show how transparent communication security policies can be implemented with existing technologies and without any modifications to the operating system kernels.

Harrison05a

[Harrison05a] W. Scott Harrison, Nadine Hanebutte, Paul Oman, and Jim Alves-Foss. The MILS architecture for a secure global information grid. *Crosstalk: The Journal of Defense Software Engineering*, 2005.

Abstract: Multiple Independent Levels of Security and Safety (MILS) is a joint research effort between academia, industry, and government to develop and implement a high-assurance, real-time architecture for embedded systems. The goal of the MILS architecture is to ensure that all system security policies are non-bypassable, evaluatable, always invoked, and tamper-proof. Using these formally proven security policies guarantees information flow control, data isolation, predictable process control, damage limitation, and resource availability. As applications are not considered trustworthy components, information flow control needs to be performed by entities external to the applications. This approach allows for the integration of legacy applications that do not necessarily have security integrated into them. Therefore, the MILS architecture creates an environment that adds safeguards to previously insecure applications, allowing the integration of possibly insecure applications into a secure environment. To accomplish this in the MILS architecture, guards are

placed between communicating entities to act as message content filters and enforce information flow control. This article discusses issues concerning design and implementation of MILS components for message routing and guarding on a secure Global Information Grid facilitating net-centric warfare and defense.

Joy06a

[Joy06a] J. Joy. A content guard for adobe portable document format (pdf). Master's thesis, Dept. of Computer Science, University of Idaho, Aug. 2006.

Abstract: The accidental release of private data is a recurring problem in modern document publishing. Two primary forms of accidental disclosure are failed redaction, where underlying text is intentionally obscured but still retrievable, and metadata leaks, where hidden metadata fields should have been stripped before publishing. Security guards have been traditionally used as a protection tool when moving data between classification domains. One of the primary functions of a guard is to act as a filter to prevent the unintended exposure of data between domains. This thesis describes a type of guard for content and is intended to help authors of documents avoid unintentionally releasing private data into the public realm. The problems of failed redaction and metadata exposure are examined through case study documents in Adobe PDF. Implementation of the content guard takes the form of a plug-in for Adobe Acrobat. The plug-in methodology proved to be successful in alerting the end user to potential exposure resulting from hidden data and is a promising path for future work.

Laude04a

[Laude04a] M. Laude. Middleware guard: A security component in the mils architecture with corba/giop. Master's thesis, Dept. of Computer Science, University of Idaho, Aug. 2004.

Abstract: The Multiple Independent Levels of Security (MILS) architecture provides a platform for high assurance applications based on separation of functionality. Communication between partitions within the MILS system is the responsibility of the MILS Message Router (MMR) and the MMR relies upon message-filtering processes called guards to enforce application-level security policies. In theory, there will be a guard for every application-level protocol used by high assurance applications in the MILS system. The MILS General Inter-ORB Protocol (GIOP) Guard is one such guard, providing application-layer filtering of Common Object Request Broker Architecture (CORBA) GIOP messages. This thesis explores the background and design of the GIOP Guard, and the implementation of a testbed for prototyping and analyzing the MILS GIOP Guard using Linux TCP/IP networking. This testbed is being used to gain insight into GIOP, build a prototypical Guard, and explore the relationship between the MILS components.

Lee02a

[Lee02a] H. Lee. *Securing Mobile Agents through Evaluation of Encrypted Functionss.* PhD thesis, Dept. of Computer Science, University of Idaho, Aug. 2002.

Abstract: Mobile agent technology is a new paradigm of distributed computing that can replace the conventional client-server model. However, it has not become popular due to some practical problems, such as security. The fact that computers have complete control over all the programs makes it very hard to protect mobile agents from untrusted hosts. In this dissertation we propose a security approach for mobile agents which protects the mobile agents from malicious hosts. Our new approach prevents privacy attacks and integrity attacks on mobile agents from malicious hosts. Many people have proposed good security approaches, but most of them do not prevent both integrity and privacy attacks. We review a few security approaches for mobile agents, discuss their weaknesses and strengths, and propose a new approach that can fix many of their problems. One interesting approach is mobile cryptography proposed by Sander and Tschudin. It encrypts mobile agents and the encrypted mobile agents are executable without decryption. Implementing mobile cryptography requires an interesting type of cryptosystem called homomorphic encryption; which allows direct computation on encrypted data, but no such homomorphic encryption schemes have been previously proposed. Our new security approach is an extension of mobile cryptography, and it removes many problems found in the original idea of mobile cryptography while preserving most of the benefits. Although the original idea of mobile cryptography allowed direct computations without decryption on encrypted mobile agents, it did not provide any practical ways of implementation due to the fact that no homomorphic encryption schemes have been published. Our approach provides a practical idea for implementing mobile cryptography by suggesting a hybrid method that mixes a function composition technique and a homomorphic encryption scheme that we have found. Like the original mobile cryptography, our approach will encrypt both code and data including state information in a way that enables direct computation on encrypted data without decryption. We believe that our approach is a viable and practical means to address security problems such as integrity and privacy attacks on mobile agents.

Lee04a

[Lee04a] Hyungjick Lee, Jim Alves-Foss, and Scott Harrison. Securing mobile agents through evaluation of encrypted functions. *Web Intelligence and Agent Systems*, 2(1):1–19, 2004.

Abstract: The mobile agent technology is a new paradigm of dis- tributed computing that can replace the conventional client-server model. However, it has not become popular due to some problems such as security. The fact that com- puters have complete control over all the programs makes it very hard to protect mobile agents from untrusted hosts. In this paper we propose a security approach for mobile agents, which

protects mobile agents from malicious hosts. Our new approach prevents privacy attacks and integrity attacks to mobile agents from malicious hosts. Many people have proposed good security approaches, but most of them do not prevent both integrity and privacy attacks. We review a few security approaches for mobile agents, discuss their weaknesses and strengths, and pro- pose a new approach that can x many of their problems. One interesting approach is mobile cryptography proposed by Sander and Tschudin. It encrypts mobile agents and the encrypted mobile agents are executable without de- cryption. Implementing mobile cryptography requires an interesting types of cryptosystem called homomorphic en- cryption scheme, which allows direct computation on en- crypted data, but none of such a homomorphic encryption scheme is known yet. Our new security approach is an extension of mobile cryptography, and it removes many problems found in the original idea of mobile cryptography while preserving most of the benets. Although the original idea of mobile cryp- tography allowed direct computations without decryptions on encrypted mobile agents, it did not provide any practi- cal ways of implementation due to the fact that no homo- morphic encryption schemes are found for their approach. Our approach provides a practical idea for implementing mobile cryptography by suggesting a hybrid method that mixes a function composition technique and a homomor- phic encryption scheme that we have found. Like the orig- inal mobile cryptography, our approach will encrypt both code and data including state information in a way that enables direct computation on encrypted data without decryption. We believe that our approach is a viable and practical means to address security problems such as in- tegrity and privacy attacks to mobile agents.

Lee04b

[Lee04b] Hyungjick Lee, Jim Alves-Foss, and W. Scott Harrison. The use of encrypted functions for mobile agent security. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, 2004.

Abstract: Mobile agent technology is a new paradigm of distributed computing that can replace the conventional client-server model. However, it has not become popular due to some problems such as security. The fact that computers have complete control over all the programs makes it very hard to protect mobile agents from untrusted hosts. In this paper we propose a security approach for mobile agents, which protects mobile agents from malicious hosts. Our new approach prevents privacy attacks and integrity attacks to mobile agents from malicious hosts. This approach is an extension of mobile cryptography, as proposed by Sander and Tschudin, and it removes many problems found in the original idea of mobile cryptography while preserving most of the benefits. Although the original idea of mobile cryptography allowed direct computations without decryptions on encrypted mobile agents, it did not provide any practical ways of implementation due to the fact that no homomorphic encryption schemes are found for their approach. Our approach provides a practical

idea for implementing mobilecryptography by suggesting a hybrid method that mixes a function composition technique and a homomorphic encryption scheme that we have found. Like the original mobile cryptography, our approach will encrypt both code and data including state information in a way that enables direct computation on encrypted data without decryption.

Malladi02a

[Malladi02a] Sreekanth Malladi, Jim Alves-Foss, and Sreenivas Malladi. What are multi-protocol guessing attacks and how to prevent them. In *Proc. 7th International Workshop on Enterprise Security*, Jun. 2002.

Abstract: A guessing attack on a security protocol is an attack where an attacker guesses a poorly chosen secret (usually a low-entropy user password) and then seeks to verify that guess using other information. Past efforts to address guessing attacks in terms of design or analysis considered only protocols executed in isolation. However, security protocols are rarely executed in isolation and reality is always a case of mixed-protocols. In this paper, we introduce new types of attacks called multi-protocol guessing attacks, which can exist when protocols are mixed. We develop a systematic procedure to analyze protocols subject to guessing attacks and use this procedure to derive some syntactic conditions to be followed, in order for a protocol to be secure against multi-protocol guessing attacks. We then use the strand space framework to prove that a protocol will remain secure, given that these conditions are followed, by modeling the conditions within the framework. We illustrate these concepts using the Mellovin and Berritt protocol (EKE) as an example.

Malladi02b

[Malladi02b] Sreekanth Malladi, J. Alves-Foss, and Robert B. Heckendorn. On preventing replay attacks on security protocols. In *Proc. International Conference on Security and Management*, pages 77–83, Jun. 2002.

Abstract: Replay attacks on security protocols have been discussed for quite some time in the literature. However, the efforts to address these attacks have been largely incomplete, lacking generality and many times in fact, proven unsuccessful. In this paper we address these issues and prove the efficacy of a simple and general scheme in defending a protocol against these attacks. We believe that our work will be particularly useful in security critical applications and to protocol analyzers that are unable to detect some or all of the attacks in this class.

Malladi02c

[Malladi02c] Sreekanth Malladi, Jim Alves-Foss, and Sreenivas Malladi. Preventing guessing attacks using fingerprint biometrics. In *Proc. International Conference on Security and Management*, Jun. 2002.

Abstract: Security protocols involving the use of poorly chosen secrets, usually low-entropy user passwords, are vulnerable to guessing attacks.

Here, a penetrator guesses a value in place of the poorly chosen secret and then tries to verify the guess using other information. In this paper we develop a new framework extending strand space theory in the context of these attacks to analyze the effect using fingerprint biometrics in those protocols. In particular, we will prove the efficacy of biometrics in preventing some known forms of guessing attacks which differ in the way the guess is verified. Interestingly, our approach shows a remarkable increase in security of selected protocols, subject to off-line guessing attacks. We illustrate these concepts on some examples.

Malladi02d

[Malladi02d] S. Malladi. A general scheme to prevent replay attacks on security protocols. Master's thesis, Dept. of Computer Science, University of Idaho, Dec. 2002.

Abstract: Replay attacks on security protocols have been discussed for quite some time in the literature. However, the efforts to address these attacks have been largely incomplete, lacking generality and many times in fact proving unsuccessful. In this thesis we address these issues and prove the efficacy of a simple and general scheme in avoiding these large class of attacks. We believe that our work will be particularly useful in security critical applications and to general protocol analyzers that are unable to detect some or all of the attacks in this class.

Malladi03a

[Malladi03a] S. Malladi and J. Alves-Foss. How to prevent type-flaw guessing attacks on password protocols. In *Proc. Foundations of Computer Security*, Jun. 2003.

Abstract: A message in a protocol is said to have a type-flaw if it was created with some intended type, but is later received and treated as a different type. A type-flaw guessing attack is an attack where a password is guessed and verified by inducing type-flaws in a protocol. Heather et al. [HLS00] prove that attacks that use type-flaws can be prevented if honest agents tag messages with their intended types. However, their tagging scheme cannot be used in a password protocol since it allows a guess to be directly verified using the tags inside password encryptions. In this paper we prove that, by following a modification of Heather el al.'s scheme, most type-flaw guessing attacks can still be prevented.

Malladi04a

[Malladi04a] S. Malladi. *Formal Analysis and Verification of Password Protocols.* PhD thesis, Dept. of Computer Science, University of Idaho, Jun. 2004.

Abstract: Cryptographic protocol analysis has been carried out vigorously over the last decade. Several papers were published which describe formal analysis techniques to analyze and verify protocols. However, password protocols (which are a special class of cryptographic protocols) remained largely ignored. The issue here is, many analysis and protection techniques for cryptographic protocols are inapplicable for password protocols, since they enable guessing attacks. In this dissertation, we present

newly found attacks called *multi-protocol guessing attacks* on password protocols which are possible in the absence of techniques to prevent protocol-interactions. We explain the development of a new constraint-based analysis tool to automatically find type-flaw and multi-protocol guessing attacks which can exist in the absence of techniques to prevent type-flaws and protocol-interactions. We formally prove that most type-flaw guessing attacks can be prevented by retaining type-tags inside all encryptions except password encryptions. We demonstrate that well-established result on preventing type-flaw attacks, cannot prevent attacks arising due to type-flaws in constructed keys and function applications such as hashing and signatures. We also show that the proof methodology adopted itself is flawed since it proves that type-tagging is effective in presence any set of inference rules. Finally, we prove that a new scheme that we call NUT is strong enough to prevent type-flaw attacks even in presence of constructed keys and function applications. Our proof avoids the flaw in previous attempts by carefully pointing out exactly why and how it is valid only under those set of inference rules which decompose a set of terms to add subterms of those terms.

Manz05a

[Manz05a] D. Manz. A network simulator for group key management algorithms. Master's thesis, Dept. of Computer Science, University of Idaho, Dec. 2005.

Abstract: The field of Group Key Management is of primary importance for securing communication amongst a group of people who have no prior shared knowledge. Often in cryptography, a secret key must be possessed by all parties who wish to communicate securely. These keys are often predetermined. However, group key algorithms enable groups to establish a shared session key without possessing predetermined keys. We created a simulation to mimic real networks in order to test and run several Group Key Management algorithms. We used Network Simulator Version 2 (NS-2) to create real-world simulations of the environments in which Group Key Management algorithms operate. This thesis describes our simulations, including the network topologies, and the actual code for the experiments. Further research into Group Key Management simulation requires a realistic evaluation of performance costs (bandwidth utilization and timing), such as is examined in this thesis. There has been little along the lines of realistic simulators of Group Key Management algorithms. Our simulator aims to provide a practical simulation of network hardware on which four Group Key Management algorithms can operate. This simulator allows us to have a realistic evaluation of four prevalent Group Key Management algorithms: EGK, TGDH, STR, and CCEGK. Additionally, a thorough discussion of the experimental results is placed in context with our theoretical predictions.

Manz07a

[Manz07a] D. Manz, J. Alves-Foss, and S. Zheng. A network simulator for group key management algorithms. *Journal Information Assurance and Security*, 2(4), 2007 in

press.

Abstract: The need for rapidly configurable, secure communication among groups of participants has resulted in the study of group key agreement protocols. The study of these protocols has been primarily theoretical. In this paper, we present the results of simulation studies of the methods provided by four group-key agreement protocols, EGK, TGDH, STR and CCEGK. The results of the simulation clarify the theoretical metrics, but also provide insight into the actual relative impact of the metrics, specifically the impact of synchronization. Overall CCEGK performed better in all categories than the other three protocols.

Meyers07a

[Meyers07a] D. Meyers. An attribute grammar for alert aggregation in intrusion detection systems. Master's thesis, Dept. of Computer Science, University of Idaho, May 2007.

Abstract: The development of a common model for describing intrusion events is one of the key issues in designing a distributed intrusion detection system (DIDS). In a DIDS, heterogeneous sensors (host-based, anomaly-based, signature-based, etc.) exchange alert messages over the network. IDMEF (Intrusion Detection Message Exchange Format) is an XML-based encoding for alerts that achieves interoperability between heterogeneous sensors, but with a severe increase in the bandwidth for message exchange. In light of the problem of alert flooding, any bandwidth overhead makes IDMEF infeasible for most DIDSs. This thesis demonstrates that reducing alerts through low-level aggregation, before message exchange, offsets the XML overhead and makes IDMEF acceptable. The proof-of-concept experiments in this thesis use Snort and the Snort-IDMEF plugin under Solaris. Four types of events are studied: port scans, SNMP requests, SQL injections, and DDoS attacks. Results show an average reduction rate for these types of alerts to be 64thesis is to provide a descriptive formalism for aggregation by means of attribute grammars for the IDMEF report language. Defining aggregation semantics through a common report language separates the formalism from the implementation and is IDS independent. In addition, the grammatical constraints are expressed in XSLT/XPath and checked by an XSLT processor, which provides access to a growing number of open source tools and resources related to XML technology.

Oman04a

[Oman04a] P. Oman, A. Krings, D. Conte de Leon, and J. Alves-Foss. Analyzing the security and survivability of real-time control systems. In *Proc. IEEE Systems, Man and Cybernetics Information Assurance Workshop*, pages 342–349, Jun. 2004.

Abstract: Many problems found in complex real-time control systems can be transformed into graph and scheduling problems, thereby inheriting a wealth of potential solutions and prior knowledge. This paper describes a transformation from a real-time control system problem into a graph theoretical formulation in order to leverage existing knowledge of graph theory back into the real world network being analyzed. We use a five-step transformation that converts an example electric power SCADA system into a graph model that allows for solutions derived from graph algorithms. Physical and logical characteristics of the SCADA system are represented within the model in a manner that permits manipulation of the network data. System vulnerabilities are identified and compared via graph algorithms prior to transformation back into the real-time control system problem space. The SCADA system analysis serves as an example of exploiting graph representations and algorithms in order to encapsulate and simplify complex problems into manageable and quantifiable models

Robinson07a

[Robinson07a] J. Robinson. A high-assurance multi-level secure file server. Master's thesis, Dept. of Computer Science, University of Idaho, Dec. 2006.

Abstract: High assurance computing systems, where the level of criticality can result in loss of life, financial disruption or other negative high-impact incidents produce a need for certifiable, verifiable and evaluatable system development process. The Common Criteria is such a certification process used to evaluate systems requiring a high level of assurance for security functionality. The Common Criteria has seven Evaluation Assurance Levels, where the amount of rigor in regards to testing, design, specification and verification increases with each level. At the highest level, formal methods are required to state a formal security policy, a high level functional model of the system and to prove the formal model satisfies the security policy. In this thesis, we demonstrate a formal model of a multi-level secure file server designed to operate within the Multiple Independent Levels of Security (MILS) architecture. The model meets the Common Criteria EAL5+ formal methods requirement and serves as an example application of the MILS architecture. The MILS architecture is enabling technology for high assurance systems such as the Global Information Grid [Har05]. The file server design incorporates concepts from existing research in multi-level file servers and relational databases, notably to promote confidentiality and integrity of information, while utilizing separation and non-interference. The model uses a virtual file system that binds together multiple single-level file systems into a virtual multi-level entity. The model is implemented in the formal methods tool ACL2 and is verified to show that it adheres to the formal specifications and security policy.

Robinson07b

[Robinson07b] J. Robinson and J. Alves-Foss. A high assurance MLS file server. *ACM Operating Systems Review*, 41(1):45–53, Jan. 2007.

Abstract: In this paper, we present the design of a high assurance file server model developed to operate within the Multiple Independent Levels of Security framework. The file server model is a multilevel application that utilizes separation to mediate information flow by adhering to a security policy formulated from a modified version of the Bell and LaPadula

Model and the GWVr2 policy, which is a separation kernel based policy developed for high assurance architectures. This paper focuses on the design aspects of the file server model and the underlying architecture. The purpose of this file server design is to develop a formal model to meet the formal methods requirement of Common Criteria, which is a system design and specification guideline for high assurance systems. The model is also an example application for the Multiple Independent Levels of Security architecture.

Robinson07c

[Robinson07c] J. Robinson, W.S. Harrison, N. Hanebutte, P. Oman, and J. Alves-Foss. Implementing middleware for content filtering and information flow control. In *Proc. Computer Security Architecture Workshop*, pages 47–53, Nov. 2007.

Abstract: This paper discusses the design and implementation of a middleware guard for purposes of content filtering and information flow control in the Multiple Independent Levels of Security (MILS) architecture. The MILS initiative is a joint research effort between academia, industry, and government to develop and implement a high assurance real-time architecture for embedded systems. The MILS architecture incorporates a separation kernel with formal system security policies that are evaluatable, non-bypassable, tamper-proof, and always invoked. Vendor specific high-level applications are assumed to be untrustworthy components; information flow control needs to be performed by middleware entities external to the applications. In the MILS architecture, a MILS Message Router and guards are placed between communicating entities to act as message content filters and enforce information flow control. As the MILS architecture does not restrict the protocols that can be employed for communications between applications, a distinct guard is needed for filtering messages within each protocol. Incorporating protocol specific guards in MILS embedded systems aids in the formal certification of those systems or the high-assurance safety critical formally-proven applications. The guards enable formally-proven security policies that guarantee information flow control, data isolation, predictable process control, damage limitation, and resource availability. An example is provided using a multi-level secure file server that uses a GIOP guard for fine-grained access control. The inclusion of a GIOP guard reduces the complexity and the effort necessary for system certification.

Rossebo06a

[Rossebo06a] B. Rossebo, P. Oman, J. Alves-Foss, R. Blue, and P. Jaszkowiak. Using Spark-Ada to model and verify a MILS message router. In *Proc. Int'l Symposium on Secure Software Enginneering*, Mar. 2006.

Abstract: The concept of information classification is used by all nations to control information distribution and access. In the United States this is referred to as Multiple Levels of Security (MLS), which includes designations for unclassified, confidential, secret, and top secret information.

The U. S. Department of Defense has traditionally implemented MLS separation via discrete physical devices, but with the transformation of military doctrine to net-centric warfare, the desire to have a single device capable of Multiple Independent Levels of Security (MILS) emerged. In this paper we present a formal model of a MILS message router using SPARK-ADA. The model is presented as a case study for the design and verification of high assurance computing systems in the presence of an underlying separation kernel. We utilized the correctness-by-design approach to secure system development and discuss the limitations of that approach for the type of system we model.

Son06a

[Son06a] J. Son and J. Alves-Foss. Covert timing channel analysis of rate monotonic real-time scheduling algorithm in mls systems. In *Proc. IEEE Information Assurance Workshop*, pages 361–368, Jun. 2006.

Abstract: The modern digital battlesphere requires the development and deployment of multi-level secure computing systems and networks. A portion of these systems will necessarily be operating under real-time processing constraints. High assurance systems processing national security information must be analyzed for possible information leakages, including covert channels. In this paper we provide a mathematical framework for examining the impact the rate-monotonic real-time scheduling algorithm has on covert timing channels. We prove that in some system configurations, it will not be possible to completely close the covert channel due to the rate-monotonic timing constraints. In addition, we propose a simple method to formulate a security metric to compare covert channels in terms of the relative amount of possible information leakage.

Son06b

[Son06b] J. Son and J. Alves-Foss. Covert timing channel capacity of rate monotonic real-time scheduling algorithm in MLS systems. In *IASTED International Conf. on Communication, Network and Information Security (CNIS 2006)*, pages 13–19, Oct. 2006.

Abstract: Real-time systems must satisfy timing constraints. In our previous work, we showed that a covert timing channel cannot be completely closed in some system configurations due to the timing constraints imposed by the Rate- Monotonic (RM) real-time scheduling algorithm. In this paper, we construct a probabilistic model to measure two quantities of a covert timing channel in RM based systems: channel capacity and quantity of specific information. We show how these two metrics can be calculated from our probabilistic model and why they are useful metrics in evaluation of a covert (timing) channel.

Son07a

[Son07a] J. Son and J. Alves-Foss. High level specification of non-interference security policies in partitioned MLS systems. In *Proc. IASTED International Conf. on Communication, Network and Information Security (CNIS 2007)*, Sep. 2007.

Abstract: We provide a formal framework for specifying the secure behaviors of a Separation Kernel (SK) with Inter-Partition Communication (IPC) capability which satisfy two requirements: 1) the Multi-Level Secure (MLS) partitioned components (called partitions) running on a SK must communicate with each other through designated communication channels, 2) IPC operations must satisfy information flow security policies such as Non-Interference. Using this framework, we present a formal model of an IPC-capable SK architecture which satisfies Non-Interference security policies.

Son08a

[Son08a] J. Son. *Covert Timing Analysis in MLS Real-Time Systems.* PhD thesis, Dept. of Computer Science, University of Idaho, Jun. 2008.

Abstract: Mathematical analysis of possible covert timing channels is a requirement for certification of high assurance Multi-Level Secure (MLS) systems. In this dissertation, we present a mathematical approach for analysis of covert timing channels in MLS real-time systems. This approach includes an analytical model which can specify the types of real-time tasks running, the real-time scheduling algorithm in use, and real-time constraints imposed on task executions. Using this analytical real-time system model, we characterize timing vulnerabilities present in real-time systems, present a methodology for measuring covert timing channel capacity, and devise countermeasures to remove or mitigate the impact of covert timing channels. Finally, we present a precise mathematical model for evaluating how performance overhead/delays of real-time systems vary with respect to the different degrees of security measures being applied.

Taylor04a

[Taylor04a] C. Taylor. *Techniques for the Survivability of Critical Computer Systems.* PhD thesis, Dept. of Computer Science, University of Idaho, Jun. 2004.

Abstract: This dissertation presents techniques developed for the survivability of critical systems from cyber based attacks. The work covers all three areas of survivability including: Attack Recognition, Attack Resistance and Attack Recovery. The dissertation is a compilation of six published, referred papers and one paper submitted for publication. Each paper targets a different aspect of the problem of critical system assurance. The techniques researched include a progressive set of solutions including: intrusion detection for Attack Recognition, software certification and formal methods for Attack Resistance and risk analysis for Attack Recovery. The research represents a broad approach to solving critical system survivability which matches the true nature of the problem. Survivability of critical systems is multi-faceted and not solvable by a single solution. While the dissertation makes a technical contribution to the security of critical systems, it will likely take contributions from many disciplines to create lasting, effective solutions to the system survivability problem.

Wahsheh06a

[Wahsheh06a] L.A. Wahsheh and J. Alves-Foss. Specifying and enforcing a multi-policy paradigm for high assurance embedded systems. *International Journal of High Speed Networks*, 15(3):315–327, Oct. 2006.

Abstract: One fundamental key to successful implementation of secure high assurance computer systems is the design and implementation of security policies. For systems enforcing multiple concurrent policies, the design and implementation is a challenging and difficult task. To simplify this task, we present an Inter-Enclave Multi-Policy (IEMP) paradigm for information access of the Multiple Independent Levels of Security and Safety (MILS) approach to high assurance system design for security- and safety-critical multi-enclave systems. The IEMP paradigm manages multiple security policies (i.e., controls the conflicts and cooperation of policies of different enclaves) within heterogeneous systems. IEMPs are policies about policies that ensure the enforcement of end-to-end mandatory information flow security policies, where the management and evolution of policies can be separated from applications. Although the approach was initially designed for use in the MILS architecture, based on the concept of a separation kernel, it is applicable to a much broader range of architectures.

Wahsheh07a

[Wahsheh07a] L.A. Wahsheh and J. Alves-Foss. Policy-based security for wireless components in high assurance systems. *Journal of Computer Science*, 3(9):727–739, 2007.

Abstract: To enable the growth of wireless networks in high assurance computer systems, it is essential to establish a security engineering methodology that provides system security managers with a procedural engineering process to develop computer security policies. Our research demonstrates how wireless communication technology is deployed using the Multiple Independent Levels of Security (MILS) architecture for high assurance computer system design of security and safety-critical multi-enclave systems to provide a framework for supporting the enforcement of diverse security multi-policies. The established wireless inter-enclave multi-policy paradigm manages multiple wireless security policies within heterogeneous systems. Applying the policy refinement rules presented in this work for a security enforcement procedure of an application system will reduce the proof effort for secure components.

Wahsheh07b

[Wahsheh07b] L.A. Wahsheh and J. Alves-Foss. Using policy enforcement graphs in a separation-based high assurance architecture. In *Proc. IEEE International Conference on Information Reuse and Integration*, pages 183–189, Aug. 2007.

Abstract: As the use of computer systems becomes more commonly employed, managing security becomes more complex. One fundamental key to effective enforcement of security standards is the support of security

policies. We present a novel graph-based approach to the specification of security policies and verification of designs that enforce the policies. This methodology provides system security managers with a procedural engineering approach that will ensure that security policy enforcement is addressed during the process of refining of the high-level system design down to a low-level implementation. We present an inter-enclave multi-policy paradigm using Policy Enforcement Graphs for information access of the Multiple Independent Levels of Security and Safety (MILS) approach to high assurance system design for security-and safety-critical multi-enclave systems. Our methodology is structured and allows for policy evolution development.

Wahsheh08a

[Wahsheh08a] L. Wahsheh. *Security Policy Design and Implementation in High Assurance Computer Systems.* PhD thesis, Dept. of Computer Science, University of Idaho, Jun. 2008.

Abstract: One fundamental key to successful implementation of secure high assurance computer systems is the design and implementation of security policies. For distributed systems enforcing multiple concurrent policies, the design of correct implementation mechanisms is a challenging and difficult task. To simplify this task, this dissertation introduces a formal security policy framework that establishes a coherent security policy assurance methodology that is efficiently and effectively applied to increase the overall security in high assurance computer systems. Although the novel methodology is designed for use in the Multiple Independent Levels of Security (MILS) architecture, a high assurance computer system design for security and safety-critical multi-enclave systems, it is applicable to a much broader range of architectures The framework includes a security policy that defines rules that regulate information access, a security model that provides a representation of the policy which enables reasoning about the system, and a security enforcement mechanisms that applies the actions imposed by the policy and stated in the model. The framework consists of fiver interrelated phases: policy specification, policy integration, policy verification, policy validation, and policy implementation. Multiple independent policies are specified as formulaes that describe relationships between sets of entities based on predicate logic with a solid mathematical foundation. These multiple policies are then integrated into a single system policy by applying an inter-enclave multi-policy classification paradigm for information access. Then,. a resolution theorem prover is used to verify system correctness with respect to policies in their life-cycle strategies. A graph-based visualization tool is then used to validate4 policies and provide system security managers with a process that enables policy reviews and visualizes interactions between systems's entities. Finally a policy implementation model id developed to securely control information access.

Wahsheh08b

[Wahsheh08b] L.A. Wahsheh and J. Alves-Foss. Security policy development: Towards a life-cycle and logic-based verification model. *American Journal of Applied Sciences*, 5(9):1117–1126, 2008.

Abstract: Although security plays a major role in the design of software systems, security requirements and policies are usually added to an already existing system, not created in conjunction with the product. As a result, there are often numerous problems with the overall design. In this paper, we discuss the relationship between software engineering, security engineering, and policy engineering and present a security policy life-cycle; an engineering methodology to policy development in high assurance computer systems. The model provides system security managers with a procedural engineering process to develop security policies. We also present an executable Prolog-based model as a formal specification and knowledge representation method using a theorem prover to verify system correctness with respect to security policies in their life-cycle stages.

Wahsheh08c

[Wahsheh08c] L.A. Wahsheh, D. Conte de Leon, and J. Alves-Foss. Formal verification and visualization of security policies. *Journal of Computers*, 3(6), 2008.

Abstract: Verified and validated security policies are essential components of high assurance computer systems. The design and implementation of security policies are fundamental processes in the development, deployment, and maintenance of such systems. In this paper, we introduce an expert system that helps with the design and implementation of security policies. We show how Prolog is used to verify system correctness with respect to policies using a theorem prover. Managing and visualizing information in high assurance computer systems are challenging tasks. To simplify these tasks, we show how a graph-based visualization tool is used to validate policies and provide system security managers with a process that enables policy reviews and visualizes interactions between the systems entities. The tool provides not only a representation of the formal model, but also its execution. The introduced executable model is a formal specification and knowledge representation method.

Wang06e

[Wang06e] B. Wang and J. Alves-Foss. An MSLS-EMM for enforcing confidentiality in malicious environments. In *IASTED International Conf. on Communication, Network and Information Security (CNIS 2006)*, pages 126–131, Oct. 2006.

Abstract: The use of security policy enforcement mechanisms has been a topic in recent literature. Particular focus has been on the class of policies that can be enforced by these mechanisms but not on the security policy guiding the execution of the monitoring mechanisms. It has been a challenge to enforce information confidentiality in a multi-level secure system since malicious users can exploit covert channels within the

enforcement mechanisms to propagate confidential information. In this paper, we characterize necessary security properties for an enforcement mechanism that can ensure secure execution of the untrusted programs even though they may be malicious.

Wang06f

[Wang06f] B. Wang. Possibilistic information flow analysis and formal verification of multiple single-level secure execution monitoring mechanisms for high assurance systems. Master's thesis, Dept. of Computer Science, University of Idaho, May 2006.

Abstract: Runtime enforcement mechanisms are needed in systems where not all system components are trusted to conform to the system security policies. Therefore the security of a system applying runtime enforcement mechanisms, to a large extent, relies on the security of those enforcement mechanisms. In this thesis we created a secure specification of a trusted enforcement mechanism. We interpret trustworthiness as 1) the mechanism satisfies the desired security requirements, 2) the security requirements do not rely on the specific implementation of the mechanism, and 3) security is formally verified using sound mathematic proofs. In order to apply mathematical strength to demonstrate the security of the mechanisms, we applied formal methods to express both the system specification and the security property. We believe that a Multiple Single Level Secure (MSLS) enforcement mechanisms could be trusted to enforce a large set of confidential security policy without compromising the underlying system security even in the circumstance where each system component are hostile.

Yang05d

[Yang05d] D. Yang. A threat-scenario-driven modeling approach to the mmr. Master's thesis, Dept. of Computer Science, University of Idaho, Aug. 2005.

Abstract: This study presents a new security modeling approach to the covert channel control security policies, a threat-scenario-driven security modeling approach. To apply this approach to a system, all potential covert channel threats are identified and modeled first; system security properties are specified based on these threat models. This approach is demonstrated through security modeling of a trusted message router, the MILS message router. Top level designs guided by the security model are proved to satisfy system security properties. The modular security modeling allows system designers to choose tradeoffs between system security and cost. The proposed modeling approach introduces two new steps at both ends of the security modeling approach. These steps smooth the transitions from abstract policies to concrete system modeling and from the formal security properties to the system design. This greatly eases the integration of security modeling into secure software development. A new definition of covert channels is proposed based on the identified gap between non-interference security and non-covert-channel security. This definition is generic. It can be used to identify covert channel threats at any system development stage of any deterministic system.

Zheng05a

[Zheng05a] Shanyu Zheng, Jim Alves-Foss, and Stephen Lee. Exploring average performance of group key management algorithms over multiple operations. In *Proc. IASTED International Conference on Communications, Internet, and Information Technology (CITT 2005)*, 2005.

Abstract: Traditional analysis of group key protocol performance is based on the cost of performing a single operation. We extend this analysis to examine the performance behavior of five group key protocols after execution of multiple operation. This paper reports the results of our experiments for 100 operations consist of combinations of join, leave, mass add and mass leave operations. The results of these experiments are consistent with the original single operation experiments, thus validating their utility.

Zheng06a

[Zheng06a] S. Zheng. *A Communication-Computation Efficient Group Key Algorithm for Large and Dynamic Groups.* PhD thesis, Dept. of Computer Science, University of Idaho, Aug. 2006.

Abstract: The management of secure communication between groups of participants requires a set of secure and efficient operations. In this dissertation we present a Communication-Computation Efficient Group Key Algorithm (CCEGK). This algorithm extends prior work to provide both efficient communication and computation, and to address performance, security and authentication issues. We then theoretically compare the performance of CCEGK with four other leading group key algorithms, EGK, TGDH, STR, and GDH3.0 in worst case scenario. Traditional analysis of group key protocol performance is based on the cost of performing a single operation. We extend this analysis to examine the performance and stability behavior of five group key protocols over multiple operations. Thus, the experimental results we report consist of combinations of join, leave, mass join, mass leave, merge, and partition operations, which assist decision makers in reviewing their performance and stability over time. Since CCEGK provides two rebalance schemes to improve its performance, the performance impact of tree rebalancing operations on the overall cost of CCEGK is evaluated over the course of the execution of multiple operations. Thus, experimental results for CCEGK obtained by executing multiple operations and considering rebalance schemes are displayed to show average computation cost, communication cost, and stability. This was done in order to provide decision makers the ability to select the appropriate protocols. Several group key protocols, including the above five, have been presented in the literature to enable secrecy of communication among dynamic groups of participants. However, there is little consistency in the literature in terms of operations supported and performance metrics of the various protocols. This makes it difficult for designers to choose the best protocol for their specific applications. To

alleviate this problem, we introduce a generic model of group key exchange protocols. In addition, we introduce new performance metrics for use in uniformly comparing these protocols and then in improving the performance of five protocols based on these operations and metrics.

Zheng06b

[Zheng06b] S. Zheng, D. Manz, J. Alves-Foss, and Y. Chen. Security and performance of group key agreement protocols. In *Proc. IASTED Networks and Communication Systems*, pages 321–327, Mar. 2006.

Abstract: A few group key protocols are analyzed, implemented and deployed, but the costs associated with them have been poorly understood. Their analysis of group key agreements performance is based on the cost of performing a single operation. In this paper we extend this analysis to examine the performance behavior of five group key protocols after execution of multiple operation. We report our experimental results for 100 operations consist of combinations of join, leave, mass join, mass leave, merge, and partition. In order to thoroughly compare the performance of five protocols, we simulate three group operations: join-leave-mass joinmass leave, merge-partition, and join-leave-mass join-mass leave-merge-partition to observe

Zheng06c

[Zheng06c] S. Zheng, J. Alves-Foss, and S. Lee. The effect of rebalancing on the performance of a group key agreement protocol. In *Annual IEEE Conference on Local Computer Networks*, pages 983–989, Nov. 2006.

Abstract: The most efficient contributory group key agreement protocols conduct their operations using a tree-based structure to guide communication and computation. Existing performance comparisons of group key protocols have only evaluated the cost of single operations in isolation. This paper expands this work by evaluating the performance impact of tree rebalancing operations on the overall cost of a group key agreement protocol over the course of the execution of several operations

Zheng07a

[Zheng07a] S. Zheng, D. Manz, and J. Alves-Foss. A communication-computation efficient group key algorithm for large and dynamic groups. *Journal of Computer Networks*, 51(1):69–93, 2007.

Abstract: The management of secure communication among groups of participants requires a set of secure and efficient operations. In this paper we extend existing work to present a CommunicationComputation Efficient Group Key Algorithm (CCEGK) designed to provide both efficient communication and computation, addressing performance, security and authentication issues of CCEGK. Additionally, we compare CCEGK with three other leading group key algorithms, EGK, TGDH, and STR. An analytical comparison of all algorithms revealed eight similar methods:

add, remove, merge, split, mass add, mass remove, initialize, and key refresh. Comparing the cost in terms of communication and computation, we found CCEGK to be more efficient across the board.

Zhou06c

[Zhou06c] Jie Zhou and Jim Alves-Foss. Architecture-based refinements for secure computer systems design. In *Proc. Policy, Security and Trust*, 2006.

Abstract: The successful design and implementation of secure systems must occur from the beginning. A component that must process data at multiple security levels is very critical and must go through additional evaluation to ensure the processing is secure. It is common practice to isolate and separate the processing of data at different levels into different components. In this paper we present architecture-based refinement techniques for the design of multi-level secure systems. We discuss what security requirements must be satisfied through the refinement process, including when separation works and when it does not. The process oriented approach will lead to verified engineering techniques for secure systems, which should greatly reduce the cost of certification of those systems.

Zhou08a

[Zhou08a] J. Zhou and J. Alves-Foss. Security policy refinement and enforcement in secure computer systems design. *Journal of Computer Security*, 16(2):107–131, 2008.

Abstract: The successful design and implementation of secure systems must occur from the beginning. A component that must process data at multiple security levels is very critical and must go through additional evaluation to ensure the processing is secure. It is common practice to isolate and separate the processing of data at different levels into different components. In this paper we present architecture-based refinement techniques for the design of multi-level secure systems. We discuss what security requirements must be satisfied through the refinement process, including when separation works and when it does not. The process oriented approach will lead to verified engineering techniques for secure systems, which should greatly reduce the cost of certification of those systems.